Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Aplikace neuronových sítí v oblasti zpracování česky psaných dokumentů

Plzeň, 2009 Martin Valenta

Originál zadání

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 1. 5. 2009 Martin Valenta

Abstract

The word of the internet is full of information nowadays. The need of sorting grows and becomes more and more important. The more data we have, the more difficult is to find the right one. The purpose of this work is to show what kind of artificial neural network we can use for categorising Czech written text documents and to find the right combination of parameters to achieve best results.

Obsah

1. Úvod

Informace jsou pro každého z nás velmi důležité. Dozvědět se odpověď na otázku snadno a rychle vede ke snaze informace shromažďovat. Dříve se informace shromažďovaly zejména v knihách, dnes se pozornost přesouvá k internetu. Na internetu může publikovat každý, což má své výhody i nevýhody. Tolik nových informací k sepsání knihy má málokdo, ale téměř každý ví něco, o co by se mohl podělit a ostatním tak ulehčit práci. Na druhou stranu zase hledat v tak obrovském množství dat, jaké je dnes na internetu, je obtížné.

Kromě kvality nás při hledání požadované informace také zajímá čas, v kterém ji jsme schopni nalézt. Je obtížné a časově náročné se probírat desítkami nerelevantních informací než najdeme tu, kterou hledáme. Je přece mnohem jednodušší vyhledávat mezi malým množstvím dat. Redukcí dat ale nesmíme ztratit ta data, která by mohla vést k odpovědi na naše otázky. Nyní vyvstává otázka, zda by informace nešlo nějak automaticky kategorizovat. Nejlépe tak, jak by to udělal člověk.

Proto, abychom mohli napodobit konání člověka, se musíme zamyslet nad tím, jak člověk danou operaci vykonává. Bezesporu by člověk ke kategorizaci použil nějakou svou znalost, tedy mozek. Mozek je ale bohužel nejsložitější a nejméně prozkoumaný lidský orgán. Na základě dosavadních znalostí o lidském mozku se můžeme snažit pouze aproximovat jeho stavbu a činnost. Víme, že se mozek skládá z neuronů a ty jsou mezi sebou propojeny, vytvářejí síť. Zkusíme tedy ke kategorizaci použít umělou neuronovou síť.

Při snaze napodobit stavbu mozkové tkáně vzniklo mnoho typů neuronových sítí s různými topologiemi. Jednotlivé typy se od sebe podstatně liší, ale základ je stejný. Mezi neurony se šíří vzruch a na různé podněty reagují neurony jinak. Aby bylo možné sítě využít pro různé účely a co nejvíce se tak přiblížit chování lidského mozku, je možné každou umělou neuronovou síť nastavit pomocí volitelných parametrů. Společně s volbou topologie je nalezení těch správných parametrů klíčové ke správné činnosti sítě.

Počítač umí sám o sobě akorát provádět aritmetické operace, neumí si sám získat hlavní myšlenku z nějakého textu, je tedy nutné převést tuto informaci do číselné podoby. Jinými slovy jde o to, vyjádřit obsah textu číselným popisem například ve formě vektoru. Když nebudeme mít výstižný popis obsahu, nemůžeme ani dokonalou neuronovou sítí nic kategorizovat.

Popisem jednotlivých slov v textovém dokumentu se zabývá bakalářská práce Lubomíra Krčmáře z roku 2007 a výsledky této práce zde budou využívány.

2. Teoretická část

2.1 Textové dokumenty v češtině

2.1.1 Problém českého jazyka

Jak již název této práce napovídá, budu se zabývat zpracováním textových dokumentů psaných v českém jazyce. Zpracování českého jazyka je v porovnání s jinými jazyky např. angličtinou výrazně náročnější. Tvrzení, že je čeština jeden z nejtěžších jazyků na světě, má své opodstatnění i v počítačovém zpracování [WIKCJ].

Při snaze dokument zařadit do nějaké kategorie je důležité vystihnout, o čem pojednává. Vodítkem k zjištění tohoto faktu může být kontext jednotlivých slov. Pro lepší pochopení uvedu jako příklad slovo prezident. Pokud se toto slovo vyskytuje v kontextu slova republika, je pravděpodobné, že se jedná o text o politice. V případě, že za slovem prezident následuje fotbalového svazu, bude se zřejmě jednat o sportovní článek. S takovýmto kontextem se již dá pracovat a považovat ho za jeden z faktorů udávajících kategorii dokumentu. Podstatné je, že slovní spojení prezident republiky a prezidentu republiky, které se liší pouze v skloňovaném pádě, nese stejnou informaci o kategorii dokumentu, nicméně při porovnání těchto řetězců počítačem dostaneme odpověď, že se řetězce liší. To je jeden z problémů českého jazyka, který by např. v angličtině nenastal. Mezi další jevy, které dělají zpracování češtiny náročné, patří časování, velmi bohatá slovní zásoba a další. Pokud umíme jednu skutečnost označit několika různými slovy, můžeme tím oživit náš psaný či mluvený projev, ale kategorizaci to jedině nesložití. Dvě různá slova, byť se stejným významem, budou brána počítačem jako různá. S časováním je problém analogický tomu se skloňováním.

Při zpracování textů v libovolném jazyce se navíc setkáváme se slovy, které pro kategorizaci nemají žádný význam. V češtině jsou těmito slovy zejména předložky, spojky, částice a citoslovce. Eliminací výše zmíněných problémů jazyka dosáhneme lepších výsledků při zpracování textů a tato fáze se nazývá předzpracování [KL07].

2.1.2 Předzpracování dokumentů

Před tím, než se budeme snažit určit kategorii dokumentu na základě kontextů jednotlivých slov, je třeba si text předzpracovat, aby byla kategorizace co nejúspěšnější popř. alespoň realizovatelná. Předzpracování se skládá z několika fází.

Nejprve se provede lexikální analýza. V této fázi se prochází postupně celý dokument a nahrazují se všechna interpunkční znaménka a bílé znaky mezerou. Vznikne tak jeden dlouhý řádek se všemi slovy původního dokumentu rozdělenými jednotným bílým znakem, mezerou.

Dále je na řadě lemmatizace. Při tomto procesu jsou všechna slova převáděna do tzv. základního tvaru, jinak také označovaného jako slovníkový tvar, tomuto tvaru se latinsky říká lemma a odtud název lemmatizace. Mimo jiné se lemmatizace využívá i pro fulltextové vyhledávání [IIR07].

Nyní tedy máme slova v základním tvaru oddělená mezerami a zbývá už jen vyřadit slova, která nám nijak nepomohou určit kategorii dokumentu. V Českém jazyce je takových slov 20 až 30 % a patří mezi ně předložky, spojky, částice a citoslovce.

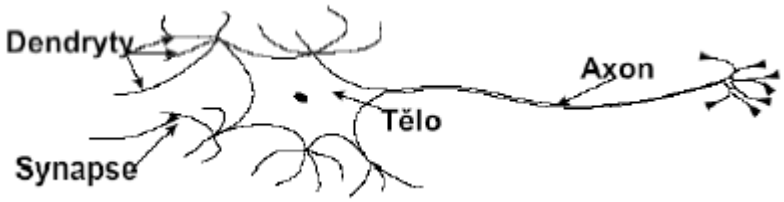
Nyní je fáze předzpracování dokončena. Kvalita výstupu závisí na kvalitě komponent provádějících jednotlivé kroky předzpracování a ovlivňuje výslednou úspěšnost kategorizace [KL07].

2.2 Od mozku k umělé neuronové síti

2.2.1 Mozek

Mozek je orgán, jemuž přisuzujeme náš intelekt, je to orgán, díky kterému jsme schopni dělat rozhodnutí, např. i takové rozhodnutí, do jaké kategorie zařadit nějaký textový dokument, proto se zde mozkem a v následujících kapitolách i jeho počítačovou aproximací – umělou neuronovou sítí - zabývám.

Mozek je řídící orgán nervové soustavy, jejímž základním stavebním kamenem je neuron. Neuron se skládá z několika částí. Mezi ty hlavní patří tělo neuronu (soma), dendrity se synapsemi a axon. Neuronů je v lidském mozku obrovské množství, přičemž velikost jejich somatu je v řádu jednotek až desítek mikrometrů. Při délce 2 až 3 mm pro dendrit je 1 m délky axonu přinejmenším pozoruhodný. Podrobnější představu o stavbě nervové buňky poskytne obr. 2.2.1.1 [ŠM04].



Obr. 2.2.1.1 Biologický neuron [HV06]

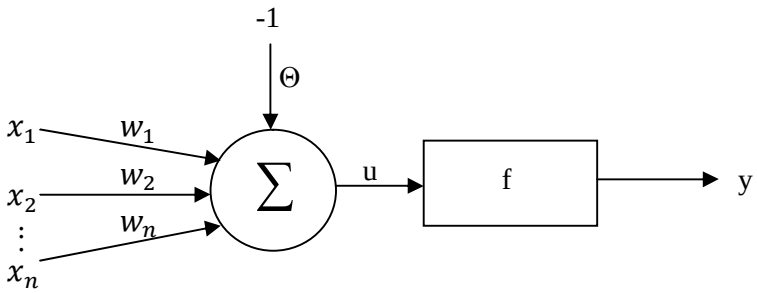
Je známo několik druhů neuronů, ale všechny mají jednu společnou funkci, a to přenos informace. Vstupním kanálem pro přenos informace jsou dendrity a výstupním kanálem je axon. Vlastní přenos informace mezi axonem vysílajícího a dendritem přijímajícího neuronu uskutečňují synapse. Dochází-li k opakovanému přenosu mezi stejnými neurony, jejich vzájemná synapse se zesiluje a můžeme tedy mluvit o učení či paměti. Každý neuron transformuje vzruch od všech svých dendritů a vyšle jej dál axonem. Tento princip přenosu informace neuronem společně s jeho stavbou jsou vzorem pro neuron v umělých neuronových sítích [KU93].

2.2.2 Umělá neuronová síť

Stejně jako biologická neuronová síť se i ta umělá skládá z neuronů, resp. modelů neuronů biologických. Snažíme se o co možná nejvěrnější podobnost mezi umělým a biologickým neuronem, ale vzhledem k faktu, že činnost mozku není lidstvem zcela probádána, můžeme hovořit pouze o aproximaci naší představy o neuronu neuronem umělým.

Při zkoumání stavby a funkce jednotlivého neuronu různých umělých neuronových sítí dojdeme k závěru, že se od sebe více či méně liší. Lze tedy říci, že se sítě dají dle typu neuronů rozdělit na sítě perceptronovského a neperceptronovského typu [ŠM04]. Sítě perceptronovského typu dle mého názoru více připomínají sítě biologické, a to zejména způsobem šíření informace a stavbou jednotlivých výkonných prvků – neuronů. Proto jako příklad umělého neuronu uvedu jednoduchý perceptron, který je základním stavebním

kamenem perceptronovských sítí a na kterém je dobře vidět, jak každý neuron přijímá na jedné své straně signály, transformuje je a výsledek vysílá jako svůj výstup. Situace je analogická situaci při přenosu vzruchu v biologických sítích. Analogie je zjevná i z porovnání schématu biologického (obr. 2.2.1.1) a umělého neuronu (obr. 2.2.2.1).



Obr. 2.2.2.1 Umělý neuron [VM]

Skrze synapse přes dendrity převede biologický neuron vzruch od jednotlivých předchůdců, s kterými je spojen, do svého těla. Vzruch přenášený jedním dendritem představuje jeden ze vstupů biologického neuronu. Vstup u perceptronu na obr. 2.2.1.2 je značen písmenem x a putuje ve směru šipky do těla. V kapitole 2.2.1 je psáno, že se síla synapsí jednotlivých spojení neuronů může měnit a tím ovlivnit vzruch nesený dendritem do těla neuronu. Tento fakt je u umělých neuronů podchycen tzv. vahami značenými písmeny w. Každý ze vstupů perceptronu je vynásoben hodnotou příslušné váhy a poté jsou výsledky sečteny, od tohoto součtu je následně odečtena hodnota parametru Θ, která slouží k potlačení vzniklého šumu. Označíme-li výsledek těchto výpočtů písmenem u, platí pro něj vztah 2.2.2.1.

$$u = \sum_{i=1}^{N} w_i \, x_i \; - \; \Theta \qquad (2.2.2.1)$$

V předchozím textu byla zmíněna jistá transformace, kterou neuron provádí a jejíž výsledek vysílá axonem. U perceptronu je tato transformace prováděna různými nelineárními funkcemi, tyto funkce se nazývají aktivační. Mezi nejběžnější aktivační funkce patří sigmoidální, gaussova a znaménková [KU93]. Konkrétních sigmoidálních funkcí je více, jejich rovnice ukazuje 2.2.2.2 [AL97].

Sigmoidální funkce $f(u) = \tanh(u)$ (2.2.2.2)

$$f(u) = \frac{1}{1 + e^{-u}}$$

$$f(u) = \arctan(u)$$

Gaussova funkce $f(u) = ae^{-\frac{(u-b)^2}{2c^2}}$ (2.2.2.3)

Znaménková funkce $f(u) = sgn(u)$ (2.2.2.4)

Výsledkem vyhodnocení aktivační funkce je výstup perceptronu, který je v obr. 2.2.1.2 značen písmenem *y*. Pokud bude spojeno více perceptronů za sebou, bude výstup n-tého vstupem n+1-ního perceptronu [KU93].

Výše popsaným způsobem funguje každý perceptron perceptronovských sítí, mezi které patří například síť MLP (Multilayer perceptron), u neperceptronovských sítí je princip podobný. Každému neuronu je přiřazen vstup, jehož transformací se vypočítá výstup neuronu. Mezi neperceptronovské sítě patří síť ART resp. SOM, jejichž popis i s konkrétními informacemi o jejich neuronech je v kapitolách 2.4 resp. 2.5 [ŠM04].

Kdyby umělé neurony pracovaly pouze tak, jak jsem doposud popsal, pak by se jejich výstup v čase nijak neměnil, tudíž by ani nemohli vykonat žádnou užitečnou činnost, nemohly se ničemu naučit a nic si pamatovat. Je tedy třeba na základě průchodu informace neuronem v minulosti měnit výstup neuronu v budoucnu, jinak řečeno dát neuronu paměť, schopnost se učit. Učení neuronových sítí je kapitola sama pro sebe a toto téma je podrobněji rozebráno v kapitole 2.3.2 [NO97].

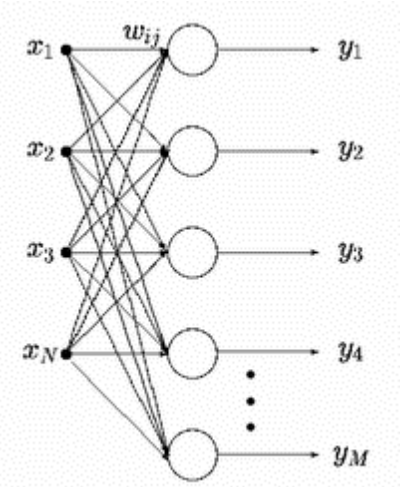
Umělé neuronové sítě jsou inspirovány sítěmi biologickými, nicméně jsou jim pouze vzdáleně podobné a to nejen proto, že činnost naší nervové soustavy nebyla dosud úplně objasněna, ale také proto, že neuronů je v mozku zhruba 10^{11} a takové množství umělých neuronů si z důvodu vysoké výpočetní náročnosti nemůžeme dovolit. Tyto důvody zapříčinily také, že vzniklo mnoho druhů sítí s odlišnými topologiemi a způsoby učení, aby bylo dosaženo nejlepších možných výsledků při řešení různých problémů [KU93].

2.3 Druhy umělých neuronových sítí

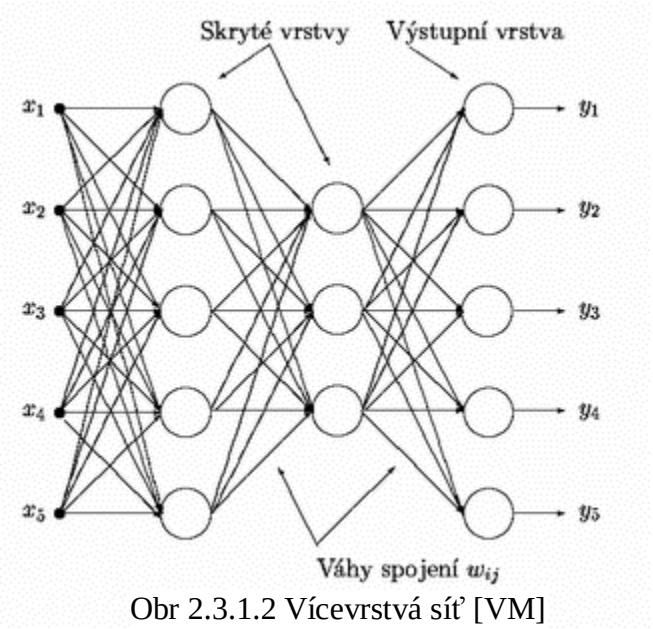
2.3.1 Dělení z hlediska topologie

Topologie udává, jakým způsobem jsou jednotlivé neurony propojeny do výsledné formace, která představuje neuronovou síť. Topologií rozumíme množinu všech neuronů a jejich propojení. V literatuře o umělých neuronových sítích se můžou jako synonymum pro topologii vyskytovat termíny architektura nebo struktura, ale já jsem se zde rozhodl používat termín topologie. Volba topologie hraje důležitou roli ve výsledném chování celé sítě a je třeba tomuto rozhodnutí věnovat náležitou pozornost [FI97].

Jednotlivé neurony umělých neuronových sítí bývají obvykle uspořádány do vrstev. Vrstvou se rozumí určitý počet neuronů vykonávajících stejnou funkci, která je určena vahami a funkcí aktivační. Po zavedení pojmu vrstva můžeme sítě rozdělit dle jejich počtu na jednovrstvé a vícevrstvé. Existují však i výjimky, kdy síť není explicitně určena jako vrstvená, nicméně takovou síť můžeme zařadit mezi jednovrstvé. Příkladem jsou asociativní paměti, konkrétně např. síť LAM [FI97]. Síť SOM se také řadí mezi jednovrstvé, zatímco ART a MLP patří k vícevrstvým. Máme-li popsat topologii sítě vícevrstvé, je nutně každou z vrstev nějak označit. Vrstva, která stojí první v pořadí zpracování vstupu, se nazývá vstupní, analogicky vrstva poslední, která dává konečnou podobu výstupu sítě, se označuje jako výstupní. Pokud síť obsahuje ještě nějaké vrstvy mezi tou vstupní a výstupní, nazýváme je skryté. Obr. 2.3.1.1 ukazuje topologii sítě jednovrstvé, zatímco obr. 2.3.1.3 sítě vícevrstvé [FA94].



Obr. 2.3.1.1 Jednovrstvá sít [VM]



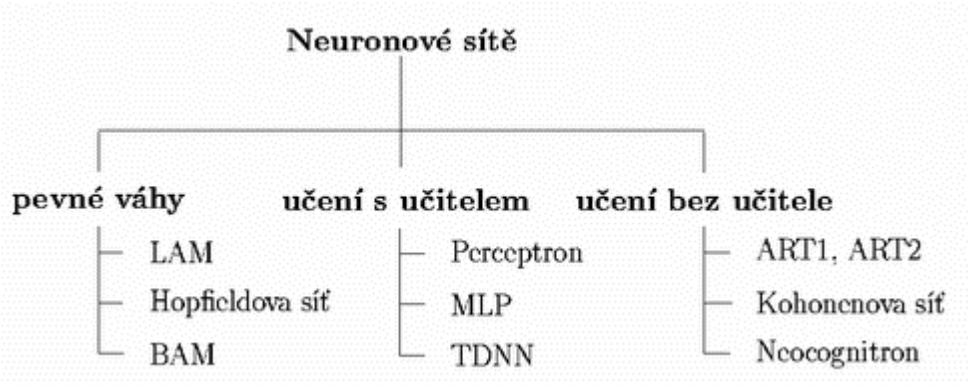
Obr 2.3.1.2 Vícevrstvá síť [VM]

Do topologie patří kromě neuronů, také jejich vzájemná propojení. Každé propojení je definováno skalární veličinou – váhou. Propojení mohou být nejen mezi neurony nacházejícími se v sousedních vrstvách, jak ukazuje obr. 2.3.1.2, ale i mezi neurony ve stejné či nesousední vrstvě. Dokonce neuron může být propojen sám se sebou [FI97].

2.3.2 Dělení z hlediska metody učení

Před tím než budeme od neuronové sítě požadovat nějakou užitečnou činnost, je třeba ji na řešení daného problému adaptovat. Biologická neuronová síť se adaptuje změnou síly jednotlivých synapsí mezi neurony. Jak už bylo řečeno v kapitole 2.2.2, sílu synapsí u umělých neuronových sítí reprezentují váhy. Hledání vhodných hodnot váhových spojení mezi umělými neurony se nazývá učení resp. trénování [KU93].

Způsobů jak dosáhnout vhodných hodnot váhových spojení je více. Jednou možností je váhová spojení nastavit fixně, tudíž hned na začátku, a jejich hodnoty již dále neměnit. Odlišný přístup zaujímá učení s učitelem a bez učitele. Rozdělení sítí dle metody učení znázorňuje obr. 2.3.2.1.



Obr. 2.3.2.1 Dělení sítí dle metody učení [VM]

Učení s učitelem vyžaduje, aby vzory, které předkládáme síti k naučení, byly doprovázeny příslušným výstupem, jenž je od sítě jako reakce na daný vstup očekáván. Výstup vyvolaný předloženým vzorem je s žádaným výstupem porovnán a spočítá se chyba, ke které došlo. Na základě této chyby se upraví jednotlivá váhová spojení tak, aby se chyba zmenšila. Naopak učení bez učitele žádnou dodatečnou informaci k učící množině vzorů nepotřebuje. Vzory jsou slučovány do shluků na základě podobnosti, kterou síť v dosavadním průběhu učení zaznamenala. Síť upravuje váhová spojení tak, aby byly stejné vzory sloučeny do společných shluků [NO97].

2.3.3 Využití sítí

Neuronové sítě obecně se využívají pro predikci, rozpoznávání, aproximaci, asociaci a další [ŠM04]. Jak již bylo zmíněno výše, volba topologie a také metody učení hraje důležitou roli ve výsledném chování sítě. Proto lze také říci, že sítě, které se hodí na řešení jednoho problému, nemusí být vhodné na řešení problému odlišného. Např. perceptron se využívá na separaci lineárně oddělitelných obrazů, MLP na klasifikaci obrazů, LAM a Hopfieldova síť se využívají na rekonstrukci šumem poškozených obrazů, ART na shlukování a SOM na vytváření sémantických map [VM]. Dále se umělé neuronové sítě využívají v medicíně, kde na základě příznaků přivedených jako vstup jsou schopné učit diagnózu, v bankovnictví, kde rozhodují o přidělení hypotéky nebo při rozpoznávání řeči [FA94].

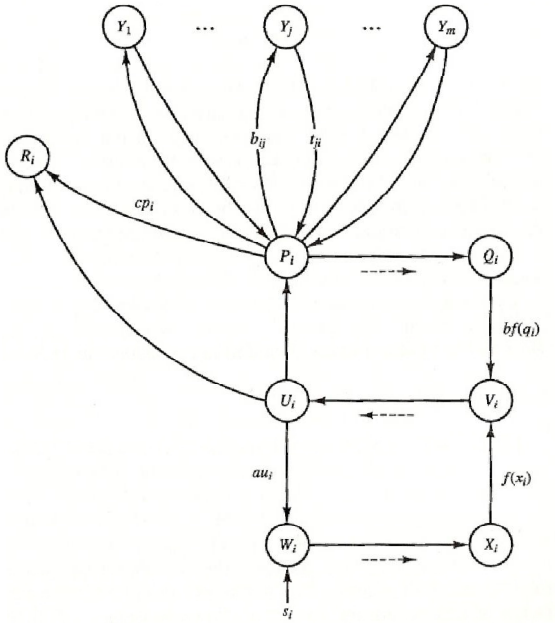
2.4 Síť ART2

2.4.1 Zařazení

Síť ART2 je jedna ze sítí typu ART. Zkratka ART znamená v překladu teorie adaptivní resonance. Mezi sítě ART patří ART1, ART2 a ART3. Zatímco síť ART1 pracuje pouze s binárními daty, ART2 pracuje s daty v podobě reálných čísel. Sítě ART jsou vícevrstvé sítě, které se učí bez učitele a hodí se zejména k rozpoznávání obrazců. U sítí učených bez učitele jsou jednotlivé předkládané vzory řazeny dle vzájemné podobnosti do shluků. Podobnost vzorů v každém shluku lze navíc u sítí ART kontrolovat. K této kontrole slouží tzv. resetovací mechanismus. Učení ART probíhá v epochách, jejichž počet si může uživatel sám zvolit. Oproti MLP je možné síť ART přiučit novému vzoru aniž by byla ohrožena stabilita celé sítě [ŠM04].

2.4.2 Topologie

Topologie ART2 se skládá ze tří druhů neuronů. První skupina neuronů zpracovává vstupy a tvoří první vrstvu sítě označovanou jako F1 vrstva. Druhá vrstva, která je pojmenována F2, je vrstva výstupní a reprezentuje shluky, které síť v průběhu učení vytvořila. Mimo vrstvy stojí již výše zmíněný resetovací mechanismus. Počet neuronů resp. shluků ve vrstvě F2 je různý, ale neurony ve vrstvě F1 mají přesně daný počet, tudíž je možné je pojmenovat. Do vrstvy F1 patří neurony označené v obr. 2.4.2.1 jako W, X, V, U, P a Q, jednotlivé shluky vrstvy F2 jsou označené písmenem Y a indexem daného shluku a resetovací mechanismus je na obrázku popsán jako R. Do topologie také patří spojení mezi neurony. Ta jsou na obr. 2.4.2.1 vyznačena šipkami [FA94].



Obr. 2.4.2.1 Topologie ART [FA94]

2.4.3. Učení

Učení sítí ART probíhá v epochách. V každé epoše jsou postupně síti prezentovány všechny vzory z učící množiny. Jednotlivé vzory jsou nejprve zpracovány vrstvou F1, kde putují tak, jak ukazují šipky na obr 2.4.2.1. Vrstva F2 je kompetitivní, tudíž všechny její neurony si spočítají odezvu na signál přicházející z vrstvy F1 a ten, který má odezvu největší se stává vítězem a tedy adeptem na naučení se předloženému vzoru. O tom, jestli se vítězný neuron bude danému vzoru učit, jinak řečeno zahrne ho jako člena svého shluku, rozhoduje resetovací mechanismus. Ten provede výpočet, který reflektuje odlišnost předloženého vzoru od ideálního reprezentanta vítězného shluku. Je-li výsledek tohoto výpočtu vyšší než uživatelem definovaná hodnota označovaná jako parametr ρ, vyvolá resetovací mechanismus reset a soutěž o nejvhodnější shluk proběhne znovu, ale už bez předchozího vítěze soutěže. Tato možnost definovat parametr ρ umožňuje uživateli kontrolovat míru shodnosti vzorů řazených do stejného shluku [FA94].

Při nalezení vítězného shluku, u kterého nedojde k resetu, je třeba upravit váhy propojení. Zde existují dva odlišné způsoby, jak to udělat. Jsou jimi tzv. rychlé a pomalé učení. Před tím, než vysvětlím oba druhy, je nutné zavést pojem rovnováha v síti. Ta nastává v okamžiku, kdy se při hledání vítěze ve vrstvě F2 neobjeví žádný reset. Při pomalém učení je váha spojení mezi výstupním neuronem vrstvy F1 a vybraným neuronem F2 upravena jednorázově a rovnováhy v síti dosaženo není, zatímco při učení rychlém jsou prováděny iterace změn vah, dokud není síť v rovnováze. Výhodou pomalého učení je, že se během jednoho předložení učícího vzoru neprovádí tak velké množství operací jako u rychlého učení. Na druhou stranu je ale třeba vykonat větší počet epoch. U rychlého učení je tomu přesně naopak, není třeba velké množství epoch, ale počet operací během jedné epochy je vyšší. Volba vhodné rychlosti učení je jedna z těch, kterou je třeba během návrhu systému brát v úvahu [FA94].

K učení sítí ART ještě zbývá dodat, že síť je citlivá na pořadí předkládaných trénovacích vzorů. Při změně pořadí trénovacích vzorů dostaneme odlišné výsledky [KU93].

2.4.4 Vybavování

Vybavování sítě probíhá téměř stejně jako učení. Vrstvě F1 předložíme vzor, který chceme zpracovat. Ve vrstvě F2 budeme hledat vítězný shluk vzhledem k výstupu z F1 a otestujeme resetovací podmínku. Nedojde-li k resetu, našli jsme vítězný neuron, v opačném případě se soutěž opakuje bez současného vítězného neuronu. Váhy propojení se už neupravují. Vyvolá-li vzor reset u všech neuronů, znamená to, že síť není pro tento vzor naučena a bude třeba vytvořit shluk nový. Vzory, jejichž vítězem se stal stejný neuron z F2, patří dle naučené sítě k sobě [FA94].

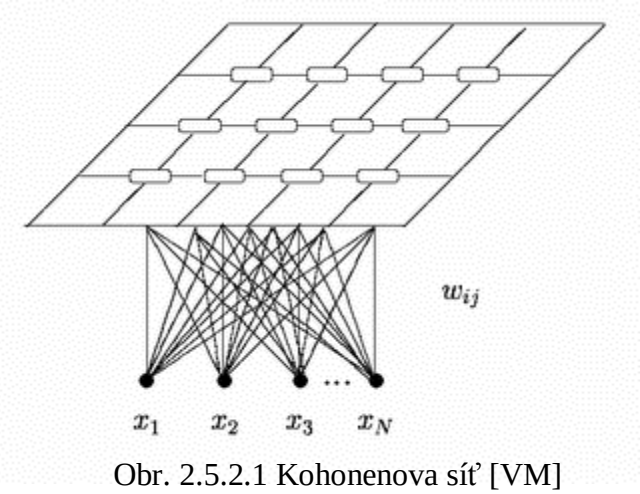
2.5 Síť SOM

2.5.1 Zařazení

Síť SOM patří mezi jednovrstvé sítě a učí se bez učitele. SOM je zkratka Self-Organising Map, což v překladu znamená samoorganizující se mapa. Principem funkce SOM je podobně jako u ART shluková analýza. Tato síť bývá také nazývána Kohonenova síť podle autora Teuvo Kohonena, který jí navrhl. Pomocí této sítě lze vizualizovat rozložení multidimenzionálních vstupních dat pomocí transformace do prostoru o nižší dimenzi [ŠM04].

2.5.2 Topologie

Podíváme-li se na množinu neuronů sítě SOM, lze rozlišit tři druhy topologií. Ve všech třech jsou neurony pravidelně uspořádány a to do jedno, dvou nebo tří rozměrných formací. Nejpoužívanější je dvourozměrné uspořádání tedy rovina. Z tohoto důvodu se síti také říká mapa. Mapa tvoří jedinou výkonnou vrstvu sítě, a proto síť patří mezi jednovrstvé. Síť umí pracovat se vstupními vektory s reálnými čísly. Jednotlivé neurony mají tolik vážených propojení, kolik dimenzí má vstupní vektor. Každá položka vstupního vektoru je tedy připojena na každý neuron sítě. Signál se od vstupního vektoru šíří směrem k síti a pak už nikam jinam, lze tedy říci, že SOM je síť s dopředným šířením [FA94].



Obr. 2.5.2.1 Kohonenova síť [VM]

2.5.3 Učení

Každý neuron sítě reprezentuje shluk, jehož vzorovým obrazem jsou váhová spojení daného neuronu. Na počátku jsou všechna váhová spojení nastavena na náhodné hodnoty.

Celá výkonná vrstva neuronů je kompetitivní, což znamená, že se všechny neurony účastní soutěže o nejlepšího reprezentanta pro právě předkládaný vstupní vektor. Metrikou pro nalezení nejlepšího reprezentanta je kvadrát euklidovské vzdálenosti jak udává vztah 2.5.3.1. Neuron s nejmenší vzdáleností se stává vítězem [FA94].

$$d = \sum_{i=1}^{N} (x_i - w_i)^2 \qquad (2.5.3.1)$$

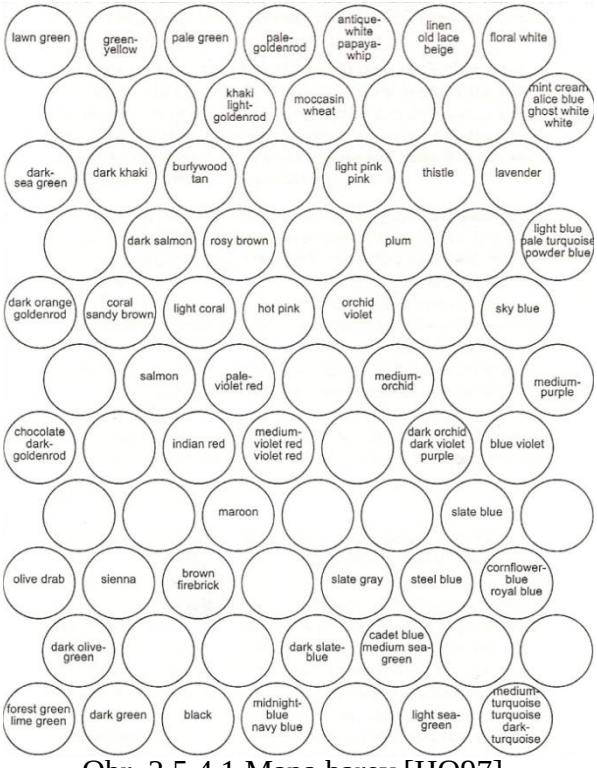
Po nalezení vítěze se upraví váhová spojení nejen vítěznému neuronu, ale i neuronům v jeho okolí. Velikost tohoto okolí se stejně jako velikost parametru α ve vztahu 2.5.3.2, který představuje učící poměr, v průběhu učení sítě zmenšuje [FA94].

$$w_{ij}(nové) = w_{ij}(staré) + \alpha \left(x_i - w_{ij}(staré)\right) \qquad (2.5.3.2)$$

Ukončovací podmínkou učení může být předložení všech vstupních vektorů, vykonání předem stanoveného počtu učících epoch nebo dosažení požadované přesnosti [ŠM04].

2.5.4 Vybavování

Po naučení sítě můžeme otestovat, jak reaguje na předložené (testovací) vstupní vektory. Každý z vektorů síti předložíme a metrikou nejmenší euklidovské vzdálenosti vybereme vítězný neuron. Tím jsme provedli transformaci z prostoru vstupních vektorů o libovolné dimenzi do prostoru s dimenzí nižší, v nejběžnějším případě do roviny. Váhy u vítězného neuronu ani jeho okolí už samozřejmě neupravujeme. Pokud si budeme zaznamenávat místa, kam byly jednotlivé vektory zařazeny, měli bychom po průchodu všech vektorů vidět podobnost mezi vektory řazenými ke stejnému neuronu. Tento výklad výsledků je stejný jako u ART, kde se také vektory považované sítí za podobné řadili ke stejnému neuronu, ale na rozdíl od ART hraje u SOM roli i okolí vítězného neuronu. Podobnost bychom tedy měli zaznamenat i u vektorů řazených do okolí vítěze. To je způsobeno tím, že jsme při učení neměnili váhy pouze neuronu vítěznému, ale i jeho sousedům. Jak dopadne zpracování RGB vektorů různých barev dvojdimenzionální sítí, ukazuje obr. 2.5.4.1 [HO97].



Obr. 2.5.4.1 Mapa barev [HO97]

2.6 Síť MLP

2.6.1 Zařazení

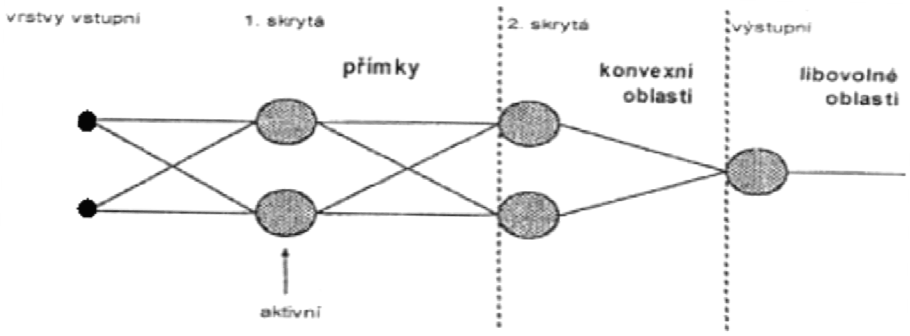
Zkratka MLP znamená v angličtině multi-layer perceptron, což je v překladu vícevrstvý perceptron. Již z názvu lze odvodit, že se jedná o síť perceptronovského typu uspořádanou do více vrstev. Z hlediska učení je MLP zástupcem sítí učených s učitelem. MLP se může učit více způsoby. Jedním z nich je algoritmus Back-Propagation, který bude popsán v kapitole 2.6.3 [ŠM04].

2.6.2 Topologie

Jak již bylo řečeno výše, síť MLP se skládá z více vrstev, ale na rozdíl od ART není jejich počet striktně definován. Každá síť MLP musí mít vrstvu vstupní a výstupní, přičemž ta vstupní pouze předává hodnoty ze vstupního vektoru dále. Velikost vstupní vrstvy je rovna počtu položek vstupního vektoru. Mezi vrstvou vstupní a výstupní se mohou vyskytovat další vrstvy, těm se pak říká skryté. U MLP je problém s určením počtu vrstev. Vzhledem k tomu, že vrstva vstupní pouze předává hodnoty dále a neobsahuje žádnou aktivační funkci, není do počtu vrstev sítě počítána. Jinak řečeno, jako počet vrstev sítě se udává počet vrstev vah [KU93].

Neurony jedné vrstvy nejsou vzájemně nijak propojené, ale každý je spojen přes váhy se všemi neurony vrstvy předchozí. Spojení jsou dobře vidět na obr. 2.3.1.2. Volba aktivační funkce je u MLP omezena. Funkce musí být spojitá a diferencovatelná [AL97].

Počet skrytých vrstev je libovolný a závisí na povaze každé úlohy, kde má být MLP nasazena. Platí ale určitá pravidla, která lze vyčíst z obr. 2.6.2.1. Tato pravidla říkají, jaké oblasti lze ohraničit při daném počtu vrstev. Např. jedním perceptronem rozdělíme rovinu na dvě poloroviny. Použijeme-li dva perceptrony v jedné vrstvě, dostáváme průnik polorovin, tedy konvexní útvar [ŠM04]. Pokud víme, jaké oblasti naše vstupní data tvoří a čím se dají oddělit, víme i počet potřebných vrstev. Zdá se, že vyšší počet vrstev znamená větší schopnost sítě od sebe oblasti dat oddělit, tudíž je lepší. V praxi to platí ale pouze do určité meze a nadměrný počet je kontraproduktivní. Pro běžné úlohy jsou tři vrstvy dostatečné [KU93].



Obr. 2.6.2.1 Ohraničující schopnosti sítě MLP [ŠM04]

Další volbou při návrhu sítě je počet neuronů každé vrstvy. U vrstvy vstupní a výstupní je to jednoduché. Vstupní vrstva má právě tolik neuronů jako je velikost vstupních dat (dimenze vstupních vektorů), zatímco u té výstupní je počet neuronů roven počtu klasifikačních tříd. Každý neuron výstupní vrstvy tedy reprezentuje jednu klasifikační třídu. U

skrytých vrstev jsou vztahy jiné, přičemž pro první v pořadí platí vztah 2.6.2.1 a pro druhou 2.6.2.2 [ŠM04].

$$N_{1.skrytá} = N_{výstupní} \left(\sqrt[3]{\frac{N_{vstupní}}{N_{výstupní}}} \right)^2 \quad (2.6.2.1)$$

$$N_{2.skrytá} = N_{výstupní} \left(\sqrt[3]{\frac{N_{vstupní}}{N_{výstupní}}} \right) \quad (2.6.2.2)$$

2.6.3 Učení

MLP patří mezi sítě učené s učitelem a zde bude popsán algoritmus Back-Propagation (dále jen BP) neboli algoritmus se zpětným šířením chyby. V tomto algoritmu probíhá učení iteračně v epochách, kdy v každé epoše předložíme síti všechny vzory z učící množiny. Fakt, že se síť učí s učitelem, si žádá, aby každý učící vzor byl doprovázen i výstupem, který od sítě na daný vstup požadujeme. Tento požadovaný výstup si lze představit jako vektor o velikosti počtu výstupních neuronů, jehož složky mají bipolární hodnoty. Chceme-li říci, že daný vektor patří klasifikační třídě, která je reprezentována i-tým neuronem, nastavíme i-tou složku vektoru požadovaných výstupů na hodnotu 1 a ostatní na hodnotu -1. Víme-li, jaké výstupy požadujeme a jaké jsme získali od sítě, lze určit chybu. Hodnota této chyby může sloužit jako ukončovací podmínka celého učení, ale lze také volit konečný počet epoch učení [FA94].

Algoritmus BP lze rozdělit do tří fází. Tou první je dopředné šíření vstupního vektoru, následuje zpětné šíření chyby a poslední fází je adaptace váhových spojení. Během dopředného šíření jsou hodnoty každého prvku vstupního vektoru skrze neurony vstupní vrstvy šířeny dále do první vrstvy skryté, zde se vypočítá hodnota aktivační funkce a šíření probíhá dále přes všechny skryté vrstvy až do vrstvy výstupní. Nyní bude využita ona dodatečná informace o žádaném výstupu sítě. Rozdíl mezi skutečným a žádaným výstupem v každém neuronu tvoří lokální chybu. Tyto hodnoty pak putují sítí zpět, čímž nastane druhá fáze učení. Při tomto zpětném chodu se také, stejně jako u dopředného šíření, využívá váhových spojení, tudíž hodnota, která putuje spojením, se vynásobí hodnotou příslušné váhy. Právě kvůli tomuto ději je pojmenován algoritmus BP. Během zpětného šíření chyby si vypočítané hodnoty v každém neuronu uchováme, jsou totiž třeba při fázi třetí, ve které se na základě chyb v každém neuronu adaptují jeho váhy. Tím je učení sítě jedním vzorem dokončeno. Chyba pro daný vstup se obvykle počítá jako kvadrát velikosti vektoru, jehož složky jsou jednotlivé lokální chyby. Tento vztah popisuje 2.6.3.1. Celková chyba je součtem chyb všech vstupních vektorů (2.6.3.2). Klesne-li celková chyba pod předem danou mez, lze učení ukončit [FA94].

Chyba jednoho učícího vzoru $$E_{jedn} = \sum_{i=0}^{M} (o_i - d_i)^2 \quad (2.6.3.1)$$

Globální chyba $$E_{celk} = \sum_{i=0}^{N} E_i \quad (2.6.3.2)$$

2.6.4 Vybavování

Po natrénování je síť připravena k užití. Činnost sítě během vybavování spočívá předložení vstupního testovacího vektoru síti a výpočtu výstupní odezvy. Jelikož každý neuron výstupní vrstvy představuje jednu klasifikační třídu, je nalezený neuron s nejvyšší odezvou výsledkem klasifikace MLP [FA94].

2.7 Aplikace neuronových sítí na zpracování přirozeného jazyka

2.7.1 Zpracování přirozeného jazyka lidským mozkem

Lidé používají k vyjádření obsahu svého vědomí písmo, řeč či gesta, přičemž písmo a řeč jsou projevem jazyka. Budeme-li zkoumat činnost lidského mozku při psaní, čtení, mluvení nebo naslouchání zjistíme, že každou z těchto činností obsluhuje jiná část mozku. Zaměříme-li se na příjem informací zrakem, bylo na základě experimentů zjištěno, že lokalita mozku, v které je vnější podnět zpracováván, také závisí na sémantice [HV06]. Domnívám se, že pokud lze rozpoznat sémantiku slova, které člověk čte na základě části mozku, která se v danou chvíli aktivuje, mohlo by to jít i u umělých neuronových sítí, jejichž výstup v sobě nese nějakou informaci o umístění. Z kapitoly 2.5 víme, že by se k tomuto účelu hodila síť SOM.

2.7.2 Zpracování přirozeného jazyka umělou neuronovou sítí

Sdělení v přirozeném jazyce jsou tvořena slovy a tato slova dávají každému jazykovému projevu jeho význam. Chceme-li tedy zpracovávat přirozený jazyk z hlediska sémantiky, musíme se zabývat zpracováním sémantiky jednotlivých slov. Při sémantickém zpracování slov je důležité odlišit stejné a podobné od různého. Nepotřebuje odlišovat slova, která nesou stejný význam, ale jsou psána jinak. Otázkou je, jakou abstrakci slov zvolit, abychom byli schopni podchytit jeho význam resp. sdružit slova s významem podobným. Odpovědí na tuto otázku se zabýval prof. E. Charniak, vedoucí laboratoře pro zpracování jazykové informace na Brown university [HO97].

Prof. Charniak definoval následující body, které je potřeba splnit, abychom získali možnost shlukovat slova s blízkým významem.

1. Určete vlastnost, která o sémantice slova vypovídá a může být nahrazena číselnou hodnotou.
2. Vytvořte vektor o délce n s číselnými hodnotami pro každý prvek, který má být zpracováván
3. Sdružte vektory, které jsou v n-dimenzionárním prostoru blízko sebe.

Tento postup říká, že hledanou abstrakcí slov by mohly být n-dimenzionální vektory, ale zavádí další problémy, které je nutné vyřešit. První z problémů se váže k bodu č. 1, který předepisuje nalezení vyčíslitelných vlastností. Jak může být něco tak symbolického jako slovo převedeno do numerické podoby? Nejjednodušší by bylo, kdybychom vzali v potaz vizuální podobu slova, ale ta ne vždy koreluje s významem slova. Podívejme se např. na slova hrabě a hrábě. Tato slova vypadají velmi podobně, ale jejich význam je naprosto odlišný. Jiným, lepším, způsobem je vytvoření tzv. kontextového vektoru. Tento způsob bere v potaz větný kontext slova, které je vektorem reprezentováno. Při tvorbě kontextového vektoru je třeba dbát na to, aby vektory představující slova s odlišným významem byly dostatečně odlišné.

Nejlépe tak odlišné, jak odlišné jsou významy slov. Konkrétním řešením kontextového vektoru může být vektor s náhodnými čísly. Při určování kontextových vektorů jednotlivých slov se do výsledného vektoru promítají vektory slov, která jsou s tím právě zpracovávaným v kontextu. Takže např. slovo předcházející a následující, ale rozsah kontextu může být i větší. Reprezentace slov pomocí kontextových vektorů má ale i své nevýhody, tou největší je, že ztrácíme informaci o různém užití téhož slova [HO97].

Druhým problémem, který nastává po přijetí postupu prof. Charniaka je, jak sdružit n-rozměrné vektory. Dle kapitoly 2.7.1 a také [HO97] se nabízí řešení pomocí dvojdimenzionální sítě SOM. Síť SOM provede žádané sdružení blízkých vektorů a tím problém převede z n-rozměrného prostoru do roviny, což je pro člověka mnohem přehlednější. Použitím sítě SOM při shlukování kontextových vektorů lze vytvořit tzv. slovní mapu. Tu získáme tak, že naučenou sítí provedeme opětovný průchod vektorů. Při tomto průchodu se pro každý kontextový vektor hledá vítězný neuron. Po jeho nalezení se slovo reprezentované daným vektorem přiřadí k vítěznému neuronu [HO97].

Výše popsaným způsobem lze pracovat se sémantikou slov, což nám dovoluje s použitím výstupu z mapy slov zpracovávat i větší jazykové celky [HO97].

2.7.3 Ohodnocení mapy slov

Dříve, než začneme mapu slov používat pro nějaké praktické účely, by bylo vhodné zjistit, jak kvalitního naučení jsme dosáhli. Jednou možností je spočítat celkovou chybu při finálním shlukování kontextových vektorů. Tento způsob klasifikace ale příliš neodráží hlavní požadavek na síť, kterým je relativní umístění shluků slov. Zejména pro testování různých nastavení sítě potřebujeme metriku, která bude brát v potaz relativní umístění jednotlivých shluků. Tuto podmínku splňuje test, při kterém nejprve vytvoříme několik seznamů slov, která by se v mapě slov měla nacházet u sebe. Poté vyhodnotíme relativní umístění nejen slov ze stejného seznamu, ale také celých seznamů. Ať už použijeme pro hledání správného nastavení sítě jakékoliv ohodnocení, je prakticky nemožné vytvořit síť, která by naplnila všechna očekávání už jen proto, že vstupní vektory neobsahují dostatečně detailní a vyváženou informaci o kontextu [HO97].

3. Realizační část

3.1 Vstupní bod této práce

3.1.1 Textové dokumenty

Tato práce se zabývá zpracováním textových dokumentů v českém jazyce. Jednotlivé dokumenty jsou zpracovávány z hlediska sémantiky a hlavním cílem je dokumenty rozdělit do kategorií dle tématu, o kterém pojednávají. Pokud by bylo možné kvalitně tematicky dokumenty rozdělit do předem vymezených kategorií, bylo by pak např. na internetu mnohem jednodušší vyhledat žádané informace.

Jako prostředek pro kategorizaci dokumentů byla zvolena umělá neuronová síť. Takových sítí ale existuje velké množství a každá nabízí nepřeberné množství kombinací svých parametrů, bez jejichž správného nastavení nelze očekávat úspěch. Aby bylo možné síť testovat a vhodné parametry nalézt, je zapotřebí co možná nejrozsáhlejšího korpusu zkušebních textových dokumentů.

Pro účely učení a testování sítí je zde využito databáze článků od ČTK. Tato databáze čítá na 7600 publicistických článků z různých kategorií. Většina jich je z oblasti sportu a politiky, ale jsou zastoupeny i jiné jako společnost, zahraniční aktualita a další. K této databázi článku také existuje jeden soubor, v kterém lze nalézt zařazení každého exempláře do kategorie popř. kategorií, pokud se daný dokument nachází někde na pomezí. Rozhodnutí o zařazení do kategorie, které je v tomto souboru uvedeno, dělal člověk a bude sloužit k porovnání s výsledky, které vzejdou ze zpracování umělou neuronovou sítí. Z celkového počtu 7600 je zhruba prvních 6000 dokumentů zařazených pouze do jedné kategorie, zbytek tvoří dokumenty smíšené, tedy někde na pomezí více kategorií. Konkrétní četnosti výskytů článků z kategorií politika, sport, zahraniční aktualita a společnost v prvních 6000 dokumentech ukazuje tabulka 3.1.1.1, na kterou bude v dalších kapitolách odkazováno.

Kategorie	Četnost
Politika	2664
Sport	3067
Zahraniční aktualita	221
Společnost	45

Tabulka 3.1.1.1 Četnosti kategorií

3.1.2 Předzpracování dokumentů

Jak již bylo zmíněno v kapitole 2.1.2, je před tvorbou kontextových vektorů a jejich zpracováním umělou neuronovou sítí třeba provést ještě několik kroků. Tato posloupnost operací se nazývá předzpracování dokumentů.

Každý článek z korpusu dokumentů získaných od ČTK již prošel fází předzpracování. Kvalita výstupu fáze předzpracování závisí na kvalitě jednotlivých použitých komponent. Při předzpracování dokument nejprve prošel lexikální analýzou, kde byly všechny interpunkční znaky nahrazeny mezerou. Toto fáze byla dle mého názoru úspěšná, protože jsem ve výsledných dokumentech žádnou interpunkci neobjevil. Dále následovala lemmatizace, jejíž úkol byl náročnější. Všechna slova se při lemmatizaci musí převést do tzv. základního neboli slovníkového tvaru. To je zjevně při bohatosti slovní zásoby českého jazyka a množství tvarů

slov, která jsme díky skloňování a časování schopni vymyslet, náročné. Nahlédnutím do dokumentů lze zjistit, že u některých slov převod do slovníkového tvaru nebyl úspěšný. Např. čtvrtý pád od slova potřeba nebyl správně převeden na pád první, tudíž v dokumentech zůstalo po lemmatizaci slovo potřebu. Další chybu, kterou jsem namátkou objevil, bylo nepřevedení slovesa v minulém čase pospíšil na infinitiv. Poslední fází předzpracování, kterou dokumenty od ČTK prošly, je odstranění nevýznamových slov.

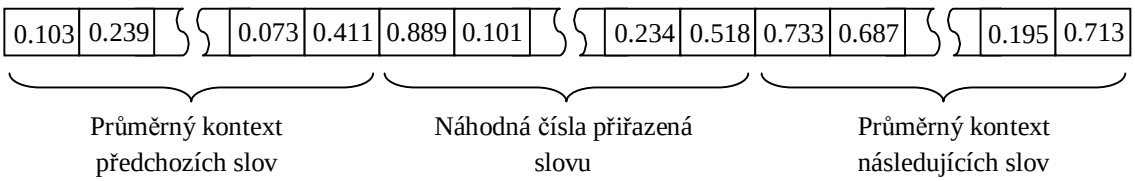
Výsledkem předzpracování všech 7600 dokumentů od ČTK je 7600 souborů, v kterých je vždy první řádka prázdná. Následuje řádka, na které je celý článek po provedení všech tří fází předzpracování a obsahuje tedy jednotlivá slova oddělená pouze mezerami. Poslední řádka každého souboru je opět prázdná.

3.1.3 Tvorba kontextových vektorů

Jako vstup mé práce slouží výstup bakalářské práce Lubomíra Krčmáře z roku 2007. L. Krčmář ve své práci využívá předzpracovaných dokumentů z databáze ČTK a zabývá se mimo jiné tvorbou slovních vektorů.

Způsob tvorby slovních vektorů prezentovaný v [KL07] vyhovuje postupu E. Charniaka, který je uveden v kapitole 2.7.2. Z databáze dokumentů ČTK je nejprve vytvořen slovník všech použitých slov i s jejich četnostmi výskytu. Dále je definována hranice minimálního a maximálního výskytu slova. Pokud se slovo v databázi vyskytuje méněkrát, než je hodnota minimální hranice nebo vícekrát než je maximální hranice slov, bude z následujícího zpracování vyřazeno. Z příliš častých nebo naopak ojedinělých slov nelze o kategorii dokumentu usuzovat.

Ke zbylé množině slov ve slovníku už je možné vytvářet kontextové vektory. Nejprve je nutné přidělit každému slovu vektor náhodných čísel. Velikost tohoto vektoru je $1/3$ velikosti kontextového vektoru. Důvod viz níže. Poté můžeme každému slovu přidělit vektor kontextový, který se skládá ze tří částí. Každá má stejnou velikost, je tedy vhodné, aby velikost vektoru byla dělitelná třemi. Při používání aplikace vytvořené L. Krčmářem v rámci jeho bakalářské práce je velikost kontextových vektorů jeden z volitelných parametrů. Prostřední část kontextového vektoru se plní hodnotami vektoru náhodných čísel danému slovu přiřazených. V počáteční resp. koncové třetině vektoru je prostor pro průměrný kontext slov předcházejících resp. následujících. Průměrný kontext předcházejících slov získáme tak, že sečteme náhodné vektory všech slov, která danému slovu předcházejí, a poté jednotlivé složky vydělíme počtem předcházejících slov. S kontextem slov následujících je situace analogická. Složení kontextového vektoru znázorňuje obr. 3.1.3.1. Jak je uvedeno v kapitole 2.7.2, je možné využít i kontext širší než jen jedno slovo předcházející a jedno následující.



Obr. 3.1.3.1 Kontextový vektor

3.2 Návrh aplikace

3.2.1 Specifikace aplikace

Mým úkolem je implementovat zvolené umělé neuronové sítě a vyzkoušet jejich schopnosti na kategorizaci textových dokumentů. Místo implementování každé sítě zvlášť jsem se rozhodl vytvořit komplexní aplikaci, která implementace jednotlivých sítí bude zastřešovat a dovolí v budoucnu jejich rozšíření o další typy. Bude pak možné jednotlivé možnosti sítí porovnat a najít tu, která bude podávat nejlepší výsledky.

Program bude sloužit k experimentálním účelům a cílem není jen najít vhodnou síť, ale také její správné nastavení. Nastavením sítě se rozumí volba volitelných parametrů, které ovlivňují výsledky zpracování. Protože nebude možné vyzkoušet veškeré dostupné kombinace nastavení parametrů, budou všechny defaultně nastaveny na hodnoty doporučené v odborné literatuře, ale uživateli bude tato hodnota zobrazena a nabídnuta ke změně.

Dále je zapotřebí, aby bylo možné předložit programu různé množiny dokumentů k učení a testování. Jedním důvodem pro tento požadavek je, že správný výběr učící množiny dokumentů může kladně ovlivnit výsledek následujícího testování kategorizace. Budeme-li předpokládat, že se numerická abstrakce dokumentů, tedy vektor, nachází v n-rozměrném prostoru a jednotlivé kategorie v tomto prostoru zabírají nějaké oblasti, je důležité, aby zvolené učící dokumenty pro všechny kategorie co nejlépe danou oblast pokrývaly. Druhým důvodem pro oddělené množiny učících a testovacích dokumentů je fakt, že používaná databáze dokumentů od ČTK neosahuje vyrovnané počty článků od všech kategorií. Jejich konkrétní četnosti ukazuje tabulka 3.1.1.1. Pokud chceme síť naučit rozpoznávat nějaký počet kategorií, měli bychom jí předložit od každé kategorie stejný počet vzorů k naučení, nicméně test kategorizace můžeme provádět na množině nevyrovnané.

Protože učení neuronových sítí je velmi časově náročné, zvláště pak při velké množině předkládaných vzorů, je vhodné, aby aplikace dávala uživateli na vědomí průběh své činnosti. Aplikace tedy bude během vykonávání práce zobrazovat okno s informacemi o tom, jaká část zpracování právě probíhá, kolik procent z dané činnosti už je hotovo, uplynulý a zbývající čas. Zbývající čas bude pouze odhadem na základě dosavadní činnosti.

Aplikace bude obsahovat grafické uživatelské rozhraní. A to zejména z důvodu co nejpohodlnějšího zadávání parametrů ke všem implementovaným typům sítí. Rozhraní bude vycházet z předpokladu, že uživatel ví, co jednotlivé parametry znamenají. Dále pak aplikace musí poskytnout detailní informace o výsledku kategorizace a to nejlépe výstupem do souboru. Ve výstupním souboru by měly být i nastavené parametry, s kterými bylo daného výsledku dosaženo, aby bylo možné zhodnotit vliv jednotlivých parametrů na úspěšnost kategorizace.

3.2.2 Výběr programovacích prostředků

Každá rozsáhlejší aplikace se nejpohodlněji tvoří ve vysokoúrovňovém jazyce, mezi které patří i jazyk Java. Tomuto jazyku jsme byli učeni již od prvního ročníku, tudíž jej ovládám nejlépe. Java nabízí programátorovi např. kolekce, které výrazně zjednodušují a urychlují vývoj aplikací. Kolekce nabízí i jiné vysokoúrovňové jazyky jako C#, ale ten jsem v počátku tvorby aplikace pro mou bakalářskou práci neuměl. Z těchto důvodů jsem si jako programovací jazyk mé práce zvolil Javu.

Vývoj aplikací v jazyce Java je možný v různých prostředích, mezi které patří Eclipse a NetBeans. Pro tvorbu práce jsem volil Eclipse, a to opět z důvodu lepší znalosti tohoto prostředí. Pro tvorbu výsledného jar souboru bude použit nástroj Ant, v kterém lze pohodlně vytvářet libovolné adresářové struktury a obsahy jar souboru.

3.2.3 Definice vstupů aplikace

Aplikace pro svou činnost potřebuje na vstupu kontextové vektory slov. Jestliže je cílem zařadit textový dokument do kategorie, je nutné mít k dispozici jak kontextové vektory všech použitých slov, tak vektory slov z každého dokumentu. Vytvoření slovníku všech použitých slov v dokumentech databáze ČTK i s jejich kontextovými vektory a vektory slov z každého dokumentu umí vytvořit aplikace vytvořená L. Krčmářem v rámci jeho Bakalářské práce. Výstup jeho aplikace slouží jako vstup té mé a bude nyní popsán.

Aplikace na tvorbu kontextových vektorů vytváří dva soubory a jeden adresář. V prvním souboru, se nachází slovník použitých slov v článcích od ČTK, ale slova nejsou všechna. Byla vyřazena ta slova, která nesplňovala svou četností podmínku na minimální či maximální výskyt. Hodnoty meze minimálního a maximálního výskytu slova se dají v aplikaci nastavit. Struktura souboru se slovníkem je následující. Na každém řádku se nachází jedno slovo slovníku. Soubor je členěn do tří sloupců, přičemž první obsahuje pořadí slova ve slovníku, v druhém je dané slovo napsáno a v třetím je uveden výskyt slova v databázi dokumentů. Tento soubor se slovníkem je pojmenovaný dle nastavených parametrů a vždy končí písmeny dic.txt, což značí, že se jedná o slovník. Druhý soubor, jehož název končí vec.txt obsahuje kontextové vektory a má právě tolik řádek, kolik jich bylo v předchozím souboru, ž čehož je patrné, že slovu na n-tém řádku v souboru se slovníkem odpovídá kontextový vektor na n-tém řádku v souboru s vektory.

Adresář, který aplikace L. Krčmáře vytvoří, je opět pojmenovaný dle nastavených parametrů, a jelikož je jen jeden, nemělo by dojít k záměně. Tento adresář obsahuje právě tolik souborů, kolik jich bylo předáno na zpracování aplikaci na tvorbu kontextových vektorů. Každému článku z databáze zde odpovídá jeden soubor. Soubory z databáze ČTK jsou pojmenovány pouze číselně a to od 1.txt do 7600.txt. Soubory ve vytvořeném adresáři mají jména stejná, jen s tím rozdílem, že za číslicí, která soubor v rámci databáze jednoznačně identifikuje, je písmeno ‘v‘. Přidané písmeno indikuje, že se jedná o soubor, v kterém byla všechna slova souboru původního nahrazena kontextovými vektory. Např. článku v souboru 123.txt odpovídá soubor 123v.txt, který v sobě obsahuje místo slov v přirozeném jazyce kontextové vektory.

Aby bylo možné ve výstupu mého programu uvést, jak byly dokumenty z různých kategorií klasifikovány, je potřeba ke každému dokumentu přidat dodatečnou informaci o příslušnosti ke kategorii. K jaké kategorii patří libovolný dokument lze zjistit ze souboru classes.info, který je součástí databáze dokumentů od ČTK.

3.2.4 Architektura aplikace

Jak naznačuje specifikace aplikace v kapitole 3.2.1, bude se jednat o vcelku rozsáhlou komplexní aplikaci s mnoha třídami. Aby se dalo v programovém kódu vyznat co nejlépe a následné úpravy či rozšíření funkčnosti nečinilo problémy, bude nutné program rozdělit. Program bude data načítat z textových souborů, zpracovávat je umělou neuronovou sítí a celá tato činnost bude ovládána pomocí grafického uživatelského rozhraní. Nabízí se tedy populární nástroj na členění kódu - třívrstvá architektura, která se skládá z datové, aplikační a prezentační vrstvy.

Nejníže položenou vrstvou je vrstva datová, ta se má starat o načítání dat z externích souborů. Do této vrstvy tedy budou zařazeny třídy, které budou načítat data z výstupních souborů aplikace na tvorbu kontextových vektorů. Nejprve je zapotřebí načíst data ze souboru, který obsahuje kontextové vektory všech použitých slov v databázi ČTK. Aby bylo možné sledovat, jak se jednotlivé vektory jako zástupci slov v neuronové síti shlukují, je vhodné ke každému kontextovému vektoru ještě přidat informaci o slovu, které vektor představuje. Bude tedy zapotřebí načítat v datové vrstvě i data z celkového slovníku. Dále bude program potřebovat kontextové vektory slov použitých v jednotlivých dokumentech, budou se tedy načítat vybrané soubory z adresáře, kde jsou obsahy dokumentů reprezentovány pomocí příslušných kontextových vektorů. Posledním úkolem datové vrstvy bude načítat data ze souboru, v kterém jsou dokumenty zařazeny do kategorií. Pro předávání načtených dat vyšší vrstvě je dle mého názoru nejlepší a nejjednodušší volit kolekce, které zvolený programovací jazyk nabízí.

Nad vrstvou datovou leží vrstva aplikační. V té se mají nacházet třídy vykonávající hlavní logiku aplikace. V případě aplikace k této bakalářské práci se v aplikační vrstvě budou nacházet implementace jednotlivých umělých neuronových sítí a třídy pro správu jejich činnosti. Rád bych každou z použitých sítí uzavřel do samostatného balíku, aby tvořila samostatný celek a byla použitelná i v jiných aplikacích. Tímto postupem by se opět zjednodušilo pozdější přidávání dalších typů sítí.

Pro testování kategorizace dokumentů bude třeba vždy kombinace dvou neuronových sítí. Z hlediska architektury aplikace je můžeme nazývat dolní a horní. První v pořadí zpracovávání vstupů je síť dolní, poté ta horní. Typ sítě dolní i horní můžeme volit libovolně, mohou být typu stejného i odlišného. Z hlediska výběru sítí je ale třeba brát v úvahu, že výrazně ovlivňuje výsledné chování a výsledky kategorizace.

Na vrcholu třívrstvé architektury leží vrstva prezentační. V této vrstvě budou všechny třídy, které budou mít na starosti zobrazování. Po spuštění aplikace se uživateli objeví hlavní okno. V tomto okně si bude moci zvolit, jakou síť chce použít, a nastavit potřebné parametry dané sítě. Bude také třeba umožnit zadávání všech vstupních souborů. Dále bude v prezentační vrstvě implementace okna, v kterém se bude zobrazovat průběh činnosti programu. Zde by mělo být napsáno, jakou činnost program právě provádí, kolik procent je z této činnosti hotových a bylo by vhodné také zobrazit uplynulý čas a odhad času zbývajícího, tak jak to bývá zvykem u komerčních aplikací.

Protože je od programu očekáváno zpracování dokumentů neuronovou sítí a zároveň zobrazování informací o průběhu činnosti, je zapotřebí využít k tomuto účelu vlákna. Vlákno pracovní bude vykonávat činnost v aplikační vrstvě, tedy učení a vybavování neuronových sítí, a druhé – informační - bude uživatele informovat o průběhu. Takto mohou obě činnosti fungovat alespoň pseudoparalelně. Protože je ale hlavním úkolem programu pracovat s umělými neuronovými sítěmi a nikoliv býti interaktivní, je vhodné uzpůsobit plánování vláken ve prospěch vlákna pracujícího.

3.2.5 Úloha spodní neuronové sítě

Spodní neuronová síť je ta, která stojí z hlediska zpracování vstupů jako první a jejím cílem je reprezentovat jednotlivé dokumenty vektory reálných čísel (tzv. dokumentovými vektory). Tvorba probíhá následovně. Nejprve se síť naučí vektory všech slov, která byla v databázi dokumentů použita. Toho lze docílit tak, že vezmeme každý vektor ze souboru, kde

jsou jednotlivá slova slovníku nahrazena kontextovým vektorem, a předložíme ho síti k naučení. Výslednou naučenou síť již můžeme využít k vytváření dokumentových vektorů.

Dokumentový vektor se vytváří na základě průchodu kontextových vektorů příslušných slovům jednoho dokumentu neuronovou sítí. Při tomto průchodu se už neprovádí učení, ale vybavování. Vybavování u sítí ART, SOM i MLP probíhá tak, že se vybere z výstupní vrstvy neuron, který na daný vstup nejvíce reaguje, tedy jeho výstup je větší než výstup ostatních neuronů z výstupní vrstvy. Takovému neuronu se říká vítěz. Budeme-li provádět průchod všech kontextových vektorů daného dokumentu naučenou sítí, můžeme postupně získat všechny nejvíce reagující neurony i s jejich odezvami. Pokud odezvy v rámci každého vítězného neuronu sečteme (jeden neuron mohl mít největší výstup na více kontextových vektorů) a následně součet vydělíme počtem kontextových vektorů obsažených v daném dokumentu, budeme mít průměrnou odezvu každého neuronu na daný dokument. Neuron, který se nestal ani jednou vítězem, bude mít průměrnou odezvu rovnu nule, zatímco neuron, který byl častým vítězem, bude mít odezvu oproti ostatním větší. Nelze ovšem říci, že nejčastější vítěz bude mít průměrnou odezvu nejvyšší, protože nezávisí jen na četnosti výher, ale také na velikosti každého výstupu při výhře. Výsledný dokumentový vektor poskládáme právě z průměrných odezev jednotlivých neuronů sítě, tudíž velikost dokumentového vektoru je rovna počtu neuronů ve výstupní vrstvě použité neuronové sítě. Dokumentový vektor je vhodné sdružit s informací o kategorii dokumentu, která bude využita při výpisu výstupu aplikace.

3.2.6 Úloha horní neuronové sítě

Horní neuronová síť zpracovává výstup sítě dolní, tedy dokumentové vektory. Dokumentovým vektorům se horní síť nejprve naučí standardním způsobem a poté bude probíhat vybavování. Množina vektorů, na základě kterých si bude síť vybavovat, může, ale nemusí být stejná s množinou trénovací. Protože bude potřeba vytvořit výpis o výsledku vybavování, je nezbytné si uchovávat informaci o tom, jaký výstup vyvolaly jednotlivé dokumentové vektory, resp. k jakým shlukům (kategoriím) byly přiřazeny.

3.2.7 Volba vhodných neuronových sítí

Pro zpracování textových dokumentů a jejich zařazení do kategorie budou třeba dvě sítě. Obě dvě budou sloužit k odlišnému úkolu a právě tento fakt si žádá zvláštní zamyšlení nad výběrem každé z nich. První síti budou během trénování předkládány kontextové vektory slov z celkového slovníku a poté bude síť vytvářet vektory dokumentové, zatímco úkol druhé je shlukování podobných dokumentů. O tom, jak moc jsou zde dva dokumenty podobné, rozhodují sítě a nikoli člověk. Pokud se ale bude rozhodnutí sítě shodovat s tím, které by učinil člověk, bude se jednat o úspěšnou klasifikaci. O úspěšnosti klasifikace velkou měrou rozhodují právě zvolené sítě, jak na pozici sítě dolní, tak horní.

Důležitost správného výběru dolní sítě spočívá v tom, že ani sebedokonalejší neuronová síť, která by se použila jako ta horní, by nedokázala od sebe odlišit dva stejné vektory mající reprezentovat dokumenty odlišných kategorií. To znamená, že pokud dolní síť zvolíme chybně a dokumentové vektory, které právě dolní síť vytváří, budou příliš podobné, nemůžeme v žádném případě očekávat od sítě horní úspěšnou klasifikaci. Hlavním požadavkem na dolní síť je, aby dokázala co nejlépe vystihnout sémantiku českých kontextů reprezentovaných kontextovými vektory a odlišit od sebe i sebemenší rozdíly. Silným nástrojem v této oblasti je síť SOM, která převádí n-rozměrný prostor kontextů do méně rozměrů, např. dvou. Síť SOM byla již na zpracování sémantiky s úspěchem použita a to je

hlavní důvod, proč bude použita i zde. Finští vědci použili právě SOM na zpracování sémantiky jejich jazyka a povedlo se jim kombinací SOM-SOM (čili SOM jako dolní i horní síť) dosáhnout zajímavých výsledků.

U výběru sítě horní je situace odlišná. My máme k dispozici omezenou množinu kategorií, do kterých chceme dokumenty zařadit. Např. síť SOM vyjadřuje relativní vzdáleností dvou shluků v síti jejich podobnost, ale to my zde nepotřebujeme. Tato znalost by mohla být užitečná při řešení jiného problému a ne při řazení dokumentů do předem definované konečné množiny klasifikačních tříd. Dala by se např. zkoumat vzdálenost různých kategorií ve výsledné mapě. Síť SOM zde zavádí dokonce navíc jeden problém a tím je, že hranice jednotlivých shluků v síti není jednoduché určit. Dokumenty budou v síti různě rozmístěné, v lepším případě v oblastech (shlucích), ale jistě se vyskytnou případy, kdy bude vektor některého z dokumentů na půli cesty od centra jednoho shluku k druhému. Musí se pak rozhodovat mezi dvěma stejně vzdálenými kategoriemi. Sítěmi, které tímto typem problému netrpí, jsou ART a MLP. Obě dvě sítě mají jednoznačně oddělené klasifikační třídy. ART je zástupcem sítí učených bez učitele, zatímco MLP se učí s učitelem. V této práci budou jako horní síť otestovány obě. Na rozdíl od SOM nabízejí tyto dvě sítě velkou variabilitu, což ale nemusí být vždy výhodou. ART má velké množství nastavitelných parametrů a najít tu správnou kombinaci si žádá zkušenosti a velké množství pokusů, MLP má sice parametrů méně, ale zase lze upravovat její topologii různým počtem vrstev a neuronů v nich. Protože dokumentové vektory obsahují položky s reálnými hodnotami, bude jako konkrétní síť typu ART použita ART2, která zpracování takových vektorů umožňuje.

Jak je psáno v kapitole 3.2.4 o architektuře aplikace, budu se snažit o implementaci jednotlivých typů sítí tak, aby byly jednoduše znovupoužitelné. Tento fakt neumožňuje jen jednoduché vyjmutí implementace sítě z aplikace k této práci a nasazení někde jinde, ale také použití téže sítě na jiném místě v rámci stejné aplikace a s jiným účelem. Může být tedy síť ART, která byla původně navržena jako horní síť, použita i jako síť dolní a přestože si nelze od kombinace sítí ART-ART (tedy ART v dolní i horní vrstvě) moc slibovat, bude otestována.

3.2.8 Definice výstupů aplikace

Účelem aplikace je testování úspěšnosti kategorizace česky psaných textových dokumentů pomocí umělých neuronových sítí. Aby bylo možné usuzovat, zda-li bylo provedené zpracování dokumentů úspěšné, je nutné mít nějakou výstupní informaci o tom, jak byly jednotlivé dokumenty klasifikovány, kromě toho je třeba dobře porozumět chování v sítích, což povede k lepšímu zpracování dokumentů. Proto bude vhodné realizovat i jiné výstupy než jen výsledek kategorizace. Protože bude výstupních souborů k jednomu testování více, budou uloženy do společného adresáře.

Prvním a nejdůležitějším výstupem bude soubor, kde se budou nacházet informace o tom, jak byly dokumenty klasifikovány. Dokument je popsán svým zařazením ke kategorii, které bylo učiněno člověkem. V horní síti ART je vytvořen předem určený počet shluků a jak bylo řečeno v kapitole 3.2.6, program si bude během vybavování ukládat informaci o tom, které dokumenty byly přiřazeny ke každému shluku. Tuto informaci program na konci ze všech shluků vypíše. Při množství 7600 dokumentů v databázi ČTK ale není vhodné vypisovat popisek (tedy skutečnou kategorii) každého dokumentu přiřazeného k příslušnému shluku. Jednou možností je spočítat, kolik dokumentů od dané kategorie bylo shluku přiřazeno, a vypsat vždy četnost a o jakou kategorii se jedná. Mě se ale jako lepší řešení zdá uvádět vždy kolik procent z celkového počtu dokumentů dané kategorie bylo každému shluku

přiřazeno. Pozor, nemyslím tím, výpis procentuálního zastoupení dokumentů z dané kategorie v rámci všech dokumentů přiřazených k jednomu shluku. Zde by se totiž negativně projevil vliv nevyváženého počtu dokumentů v kategoriích v testovací množině, což je právě problém databáze od ČTK. Při použití MLP jako horní sítě je situace podobná. Počet klasifikačních tříd určuje počet neuronů ve výstupní vrstvě MLP a lze říci, že jeden neuron v této vrstvě představuje jednu kategorii. Je-li dokument na základě nejvyšší odezvy přiřazen některému neuronu, můžeme si během vybavování pamatovat, z jaké kategorie tento dokument byl. Po dokončení vybavování se vypíše obsah neuronů z výstupní vrstvy tak, jako u shluků v síti ART. Jediným rozdílem je, že u MLP mohou být výstupní neurony přímo pojmenovány kategorií, kterou mají reprezentovat, protože se jedná o síť učenou s učitelem a je předem definováno, který neuron má mít nejvyšší odezvu při dané kategorii. Z výstupního souboru pak bude patrné, že např. jako politika bylo klasifikováno 90 % politických a 5 % sportovních článků, zatímco u zpracování ART bude pouze vidět, že 90 % politiky a 5 % sportu bylo zařazeno do stejného shluku.

V kapitole 3.2.7 je psáno, že na způsobu tvorby a výsledné podobnosti dokumentových vektorů velkou měrou závisí úspěšnost celého zpracování. Nebude-li obsah souboru s výsledky zpracování zcela odpovídat našim představám, bylo by žádoucí mít možnost vidět, jakou podobu měly vektory klasifikovaných dokumentů. Budou-li vektory už na první pohled příliš podobné, nebo dokonce stejné, nemůžeme se divit špatným výsledkům klasifikace. Není tedy nic jednoduššího, než že se po průchodu dokumentů dolní sítí výsledné dokumentové vektory vytisknou do souboru.

Jako dolní síť aplikace byla mimo jiné zvolena i síť ART, konkrétně ART2. Není mi známo, že by tato síť byla někdy použita na zpracování sémantiky přirozeného jazyka, ale když už zde bude testována, tak by bylo vhodné sledovat, jak si s kontextovými vektory jednotlivých slov poradí. Jak bylo popsáno v kapitole 3.2.7, tato síť netrpí problémem obtížného oddělení shluků jako SOM, z čehož vyplývá, že je možné jednoduše vytvořit výpis slov všech vytvořených shluků. Protože slov ve slovníku k databázi ČTK je více než 65 tisíc, vidím jako nejlepší volbu výpis obsahu každého shluku do samostatného textového souboru. Analýzou vytvořených výpisů bude po provedení testů možné zhodnotit, jestli je použití sítě ART vhodné pro zpracování sémantiky prostřednictvím dolní sítě či nikoli.

Učení s učitelem, které se používá u MLP, umožňuje určit během učení chybu. Vývoj této chyby v průběhu konání učících cyklů lze uchovat a na závěr jej vytisknout do souboru. Ze získaných dat je např. možné sestrojit graf, jehož hodnoty budou v ideálním případě klesat k nule. Dále bych chtěl pro lepší sledování činnosti sítě vypisovat do souboru i vývoj odezvy sítě na všechny vektory z učící množiny. Myslím si, že u sítě učené s učitelem má tento výstup smysl, protože předem víme, jaké hodnoty výstupu na dané učící vektory chceme a můžeme tedy sledovat, jak a jestli se výstupní hodnoty žádaným hodnotám blíží či nikoliv. Zároveň je také možné spočítat, jestli je výstup sítě na daný vstupní dokumentový vektor lepší než v předchozím učícím cyklu. Z informací o žádaném výstupu, skutečném výstupu, kategorii dokumentu a indikátoru lepšího nebo horšího výstupu než v předchozím cyklu bych poskládal záznamy a ty pak tiskl do dalšího výstupního souboru. Je třeba mít ale na vědomí, že s rostoucím počtem učících epoch a učících dokumentů (dokumentových vektorů) bude velikost výsledného výstupního souboru růst. Bude tudíž vhodné dát tento výstup jako volitelný a tím nechat o jeho tvorbě rozhodnout uživatele. Počet učících epoch se může počítat i ve statisících a při databázi se 7600 dokumenty je možné ve výsledném souboru s vývojem výstupů sítě dosáhnout i desítek milionů řádek.

3.3 Implementace

Aplikaci k této bakalářské práci jsem pojmenoval Categorizator podle jejího účelu. Názvy tříd, proměnných a texty komentářů jsou psány v angličtině. Program je dělen do balíků, kde každá z vrstev třívrstvé architektury má svůj vlastní balík pojmenovaný dle názvu příslušné vrstvy, jen třída *Main* stojí vně. Implementace sítí ART, SOM a MLP mají taky své vlastní balíky, které z hlediska architektury aplikace náleží vrstvě aplikační.

3.3.1 Třída Main

Třída *Main*, jak už název napovídá, je tou třídou, která obsahuje statickou metodu *main*. V metodě *main* celá aplikace startuje. Jelikož je program ovládán pomocí grafického uživatelského rozhraní, je úkolem metody *main* pouze toto rozhraní vyvolat, tedy vytvořit instanci hlavního okna aplikace. Během testování kategorizace jsem došel k závěru, že by bylo vhodné do této třídy doimplementovat takovou funkčnost, aby bylo možné spouštět testy bez nutnosti využívání grafického rozhraní, tedy pouze z příkazové řádky. Pro uživatele, který už program zná, by to přineslo jednodušší zadávání více testů najednou např. prostřednictvím dávkových souborů.

3.3.2 Třída Input

Tato třída je jedinou třídou datové vrstvy, má tedy na starosti veškeré načítání vstupů z externích souborů. Vstupní soubory jsou pouze textové a všechna čtení probíhají pomocí třídy *Scanner*. Třída má tři veřejné metody a všechny vracejí načtená data v kolekcích. První metodou je *getDictionary*, která na základě názvů souboru s celkovým slovníkem a souboru s kontextovými vektory ke slovům ve slovníku předaných jako parametry načte všechna použitá slova v databázi dokumentů a jim příslušné kontextové vektory. Slovo ze slovníku a jeho kontextový vektor spolu vytvoří instanci třídy *Pattern* a ta se uloží do kolekce *ArrayList*. Reference na *ArrayList* naplněný všemi načtenými daty se vrací jako návratová hodnota. Další metoda má název *getDocument* a jejím parametrem je objekt třídy *File*, což je dokument, jehož slova byla nahrazena kontextovými vektory. Metoda *getDocument* načte kontextové vektory do kolekce *ArrayList* a ten vrátí jako svou návratovou hodnotu. Poslední metodou je *getRealCategories* a ta načítá se souboru, jehož název je parametrem, ke každému dokumentu z databáze jeho skutečnou kategorii. Skutečnou se myslí tu, kterou určil člověk a ne neuronová síť. Načtená data se vrací v *HashMapě*, kde je klíčem název souboru a hodnotou jeho skutečná kategorie.

3.3.3 Třída ApplicationLogic

Třída *ApplicationLogic* už patří do aplikační vrstvy. Metody této třídy jsou volány z uživatelského rozhraní s požadavkem na konkrétní zpracování dokumentů. Protože aplikace nyní nabízí tři kombinace dolních a horních sítí (ART-ART, SOM-ART a SOM-MLP), obsahuje i tato třída tři metody. Jejich jména jsou *runART_ART*, *runSOM_ART* a *runSOM_MLP*. Všechny metody jsou volány s parametrem, kterým je objekt třídy *ParametersComplete*. Objekt této třídy v sobě uchovává všechny parametry pro kombinaci obou sítí nastavené v GUI. Každá z výše jmenovaných metod poté vykoná potřebné kroky ke spuštění zpracovávání příslušnou dvojicí sítí.

3.3.4 Třída ParametersComplete

Chování neuronových sítí se dá ovlivnit mnoha parametry, např. v případě sítě ART jich je až 12. Vzhledem k tomu, že se pro zpracování používají sítě dvě, se tedy můžeme dostat na vysoké číslo. Vysoké ve smyslu předávaných parametrů mezi metodami. Zvolil jsem tedy jiný přístup, kdy jsou parametry sítí zapouzdřeny do dvoustupňové hierarchie objektů. Objekt třídy *ParametersComplete* v sobě nese názvy všech vstupních souborů a také odkazy na objekty, které obsahují konkrétní hodnoty parametrů pro dolní resp. horní síť. Mezi metody této třídy patří akorát getry a setry.

3.3.5 Třída Net

Všechny tři dosud implementované kombinace sítí pro klasifikaci dokumentů mají hodně společného. Všechny zpracovávají nejprve slovník, poté dokumenty. Zpracování se u jednotlivých kombinací liší způsobem, ale vždy musí být vykonáno. Proto je v aplikaci třída *Net*, která je abstraktní a bude používána jako rodičovská třída konkrétních implementovaných soustav sítí. *Net* má abstraktní metodu *workOut*, která předepisuje potomkům povinnost tuto metodu překrýt a implementovat tak vlastní způsob zpracování. Schéma znázorňující činnost metod *workOut* u všech tříd, které od *Net* dědí, lze nalézt v přílohách jako schéma 2. Mezi další společné činnosti všech kombinací sítí patří výpis výsledků kategorizace a výpis dokumentových vektorů. Tyto činnosti jsou pro všechny soustavy sítí naprosto stejné, tudíž metody, které výpisy provádějí, nejsou abstraktní, ale jsou již plně implementovány a potomci je pouze volají. Výpis výsledků provádí metoda *printCategories* a výpis dokumentových vektorů metoda *printDocuments*. Aplikace má dle kapitoly 3.2.1 vypisovat i parametry, s kterými bylo výsledků dosaženo. Tato informace se nejlépe hodí právě do souboru s výsledky klasifikace a bude tedy tvořit jeho hlavičku. Výpis hlavičky s parametry se ale liší u každé soustavy sítí, a proto je realizován pomocí abstraktní metody *printCaption*, v které si každá soustava vypíše hlavičku dle sebe a až poté se tiskne společná část metodou *printCategories*. Po výpisu společné části se ještě volá abstraktní metoda *printFooter*, kde může každá soustava vypsat ještě dodatečné informace.

3.3.6 Třída ART_ART

ART_ART je jednou ze tříd, které od třídy *Net* dědí. Tato třída reprezentuje zpracování soustavou sítí, kde ART je jak dolní tak horní síť. Od třídy *Net* má předepsáno, že musí implementovat metodu *workOut*, v které jsou volány metody na učení dolní sítě slovníkem všech použitých slov, vytvoření dokumentových vektorů, učení horní sítě dokumentovými vektory a klasifikace dokumentů horní sítí. Dále musí implementovat *printCaption* resp. *printFooter*, kde se tisknou všechny parametry dolní i horní ART sítě resp. obsah shluku, kam byly zařazeny dokumenty, které u všech jiných shluků vyvolaly reset. Poslední veřejnou metodou je *printWords*. V této metodě se za každý shluk tisknou slova příslušných kontextových vektorů, které byly k danému shluku přiřazeny.

3.3.7 Třída SOM_ART

SOM_ART je další třídou, která dědí od třídy *Net*. V její veřejné metodě *workOut* se nejprve naučí dolní síť, kterou zde představuje SOM, celkovým slovníkem. Poté se pomocí dolní sítě vytvoří dokumentové vektory dokumentů z učící množiny, které slouží jako učící množina vektorů sítě horní (zde ART). Po naučení horní sítě se vytvoří dokumentové vektory

dokumentů, které mají být klasifikovány, a ty se zpracují sítí ART. V metodě *printCaption* a *printFooter* se provádí obdobné činnosti jako u metod se stejným jménem ve třídě *ART_ART*.

3.3.8 Třída SOM_MLP

Tato třída stejně jako třídy *ART_ART* a *SOM_MLP* dědí od *Net* a reprezentuje zpracování dokumentů dvojicí sítí SOM a MLP. V metodě *workOut* se provádí stejné kroky jako u SOM_ART popsané v kapitole 3.3.7, ale místo učení a klasifikace sítí ART je zde MLP. Metoda *printCaption* opět vypisuje hlavičku souboru s výsledky, ale podoba výsledků klasifikace je teď trochu odlišná, změnily se názvy shluků. Zatímco u ostatních kombinací sítí, kde je horní sítí ART, jsou shluky jen očíslovány, jsou u kombinace SOM-MLP shluky popsány názvem kategorie, jejíž dokumenty se mají ve shluku shromáždit. Protože u sítě MLP víme, které dokumenty byly zařazeny špatně a které dobře, lze určit, u kolika procent dokumentů byla klasifikace úspěšná. Tuto informaci vypisuje metoda *printFooter*. Vývoj chyby v průběhu konání epoch učení vypisuje do souboru metoda *printErrors*.

3.3.9 Třída ART2

Implementace sítě ART2 je v balíku *art2*, kde se také nachází třída *ART2*. Objekt této třídy je referencován jako reprezentace celé sítě. Veřejná metoda *buildNet* slouží k sestavení sítě z obou jejích vrstev. Mezi další metody třídy patří *learn*, která jak už název napovídá, učí síť vektory z předané kolekce *ArrayList*. Průběh učení znázorňuje algoritmus 1 v sekci přílohy. Metody *getCategorizedDescriptions* a *getAverageSignal* provádějí klasifikaci (viz algoritmus 2 v přílohách), první jmenovaná vrací ke každému shluku seznam popisků (skutečných kategorií) přiřazených dokumentů, zatímco *getAverageSignal* vrací pole o velikosti počtu shluků, ve kterém jsou jednotlivé prvky průměrnými vyvolanými odezvami na kontextové vektory předloženého dokumentu. Metodou *getAverageSignal* se tedy vytváří dokumentové vektory.

3.3.10 Třídy ART2Parameters a ART2Constants

Síť ART2 má mnoho nastavitelných parametrů, které ovlivňují její činnost při učení a vybavování. Protože by bylo krajně nepřehledné předávat každý parametr sítě prostřednictvím parametrů, s kterými jsou volány metody, rozhodl jsem se vytvořit třídu *ART2Parameters*. Tato třída má jako své atributy právě parametry sítě ART2 a její instance tedy slouží k předávání všech parametrů najednou pomocí jedné reference. Tímto způsobem se zabrání, aby např. konstruktor třídy *ART2* byl volán s více jak deseti parametry. Aby uživatel nemusel vyplňovat všechny parametry, i když jim třeba nerozumí, je tu třída *ART2Constants*. V této třídě jsou veřejnými statickými atributy všechny parametry sítě i se svou doporučenou hodnotou. Doporučené hodnoty jsou brány z [FA94].

3.3.11 Třída F1Layer

Třída F1Layer představuje jednu ze dvou vrstev sítě ART2. V pořadí zpracovávání vstupů stojí jako první, dá se tedy říci, že je vstupní. Vrstva F1 obsahuje šest typů neuronů, které jsou dle [FA94] pojmenovány *w*, *x*, *v*, *u*, *p*, *q* a tvoří atributy třídy *F1Layer*. Neurony typu *w* přebírají hodnoty ze vstupního souboru, zatímco neurony *p* předávají výstup vrstvy F1 vrstvě F2. Mezi veřejné metody patří metoda *restart*, která nastaví výstupy všech neuronů na

nulu. Této metody se využívá, při přechodu na zpracování dalšího vstupního vektoru. Metody *updateF1Activations* a *UPHalfTurn* zajišťují šíření vzruchu ve vrstvě F1.

3.3.12 Třída SetOfNeurons

Neurony *w, x, v, u, p, q* z vrstvy F1 sítě ART2 zpracovávají vstupní vektor a výsledek předávají vrstvě F2. Tyto neurony jsou implementovány jako objekty třídy *SetOfNeurons*. Během zpracování se provádějí různé aritmetické operace, které jsou popsány v [FA94]. Činnostmi, které provádějí neurony všech výše zmíněných typů, jsou předávání normalizovaného vektoru výstupů a potlačení šumu u výstupních hodnot. Metodou na výpočet normalizovaného výstupu je *getNormalizedValues*, zatímco potlačení šumu provádí metoda *getSuppressedValues*. Další operace prováděné při přenosu mezi neurony jsou implementovány ve třídě *F1Layer*, protože už nejsou pro všechny neurony společné.

3.3.13 Třída F2Layer

Shluky jsou v síti ART2 reprezentovány neurony, které se nacházejí ve vrstvě F2. Vrstva F2 je implementována v třídě *F2Layer*. Vítězný shluk se ke každému vstupu hledá pomocí metody *findAWinner*, která pro každý neuron vrstvy F2 spočítá hodnotu výstupu a mezi těmi, které nejsou zablokovány, najde neuron s odezvou nejvyšší. Metoda *inhibitCluster* zablokuje neuron, jenž je určen parametrem, s kterým je metoda volána. Po dokončení zpracování jednoho vstupního vektoru je třeba zablokované neurony zase odblokovat. Tuto činnost provádí metoda *restart*. Pro tvorbu dokumentových vektorů je třeba získat vektor s průměrnou odezvou sítě. Tento vektor vrací metoda *getAverageSignal*. Metoda *getDescriptions* se volá po klasifikaci a vrací ke každému neuronu vrstvy seznam popisků dokumentů, které k němu byly přiřazeny.

3.3.14 Třída Cluster

Neurony ve vrstvě F2, které představují shluky, jsou objekty třídy *Cluster*. Mezi atributy této třídy patří pole dopředných a zpětných vah, kolekce vektorů i s popisky, které byly shluku přiřazeny, součet signálů vyvolaných přiřazenými vektory a další. Neuron musí být schopen vypočítat svou odezvu na předložený vektor, což vykonává metoda *getSignal*. Přidání vektoru s popiskem do vítězného shluku je realizováno přidáním do kolekce členů shluku, která se jmenuje *members*. Vlastní přidání provede metoda *setWinnerOf*. Metoda *getAverageSignal* vrátí průměrnou odezvu neuronu na vektory předkládané síti. Hodnota průměrné odezvy se vypočítá jako podíl součtu signálů vyvolaných vektory, pro které se stal daný shluk vítězným, a celkového počtu vektorů předložených síti. Tímto způsobem jsem se snažil docílit toho, aby průměrná odezva neuronu, který se stal vítězem častěji než ostatní neurony, byla vyšší, protože shluk jím reprezentovaný bude zjevně pro předložený dokument charakterističtější.

3.3.15 Třídy SomMap a Neuron

Síť SOM je implementována ve třídě *SomMap* a jednotlivé její neurony jsou instancemi třídy *Neuron*. Obě tyto třídy jsou v balíku *som* v aplikační vrstvě. Implementaci sítě SOM jsem převzal z bakalářské práce Jiřího Kůsy z roku 2007, která se zabývá sítí WEBSOM. K bakalářské práci Jiřího Kůsy byla vytvořena celá aplikace s grafickým uživatelským rozhraním, která se sítěmi SOM pracovala. Protože já ve své aplikaci potřebuji

pouze implementaci SOM, vyjmul jsem z aplikace Jiřího Kůsy právě třídy *SomMap* a *Neuron*. Objekty třídy *Neuron* jsou referencovány ze *SomMap* a má část aplikace je tedy přímo nevyužívá. Objekt *SomMap* je odkazován ze třídy *SOM_ART* a také *SOM_MLP*, kde jsou volány metody *startTraining* pro naučení sítě a *startCategorization* pro klasifikaci.

3.3.16 Třídy MLP a ErrorRecord

MLP patří do balíku *mlp*, ve kterém se nachází všechny třídy implementace sítě MLP. Objekt třídy *MLP* je odkazován ze *SOM_MLP* jako reprezentace sítě MLP. Stejně jako ostatní sítě je i síť MLP třeba nejprve naučit vstupním vektorům. Učení znázorněné v přílohách algoritmem 3 provádí metoda *learn*. Metoda *categorize*, jejíž algoritmus je v přílohách jako algoritmus 4, má na starosti vybavování sítě MLP. V kapitole 3.2.8 je psáno, že se u zpracování sítí MLP bude vypisovat vývoj chyb a výstupů během fáze učení. Způsob výpisu chyb je jednoduchý. Po každé učící epoše se celková chyba za všechny učící vektory uloží do kolekce v podobě objektu třídy *ErrorRecord*, jenž spojuje informaci o pořadí epochy a hodnotě chyby, a na závěr se obsah kolekce vytiskne do souboru. U výpisu vývoje výstupů sítě je problém složitější. Informaci o tom, jestli je výstup blíže tomu žádanému, jsem rozdělil na části. Výstup sítě je skupina tolika čísel, kolik je výstupních neuronů. Mohou tedy tvořit vektor, který budu nazývat výstupní. Nositelem informace o lepším či horším výstupu je také vektor, kterému lze říkat hodnotící a který je tentokrát složený z písmen. Je-li i-tý prvek výstupního vektoru blíže i-témů prvku žádaného výstupu než tomu bylo u předchozího učícího cyklu, pak je i-tý prvek hodnotícího vektoru písmeno B jako better (lepší), v jiném případě bude uloženo písmeno W jako worst (horší). Takto vytvořené hodnotící vektory budou spolu s konkrétními výstupy součástí jednoho z výstupních souborů.

3.3.17 Třídy MLPParameters a MLPConstants

Chování sítě MLP se dá ovlivnit dvěma parametry. Prvním je parametr α, kterému se také říká učící poměr. Druhým je tzv. parametr potlačování šumu označovaný Θ. Oba tvoří atributy třídy *MLPParameters*, mezi které také patří počet epoch učení, maximální hodnota chyby a pole celých čísel s názvem *hiddenLayers*, které udává kolik skrytých vrstev s kolika neurony má síť mít. Počet skrytých vrstev je dán délkou pole *hiddenLayers* a počet neuronů i-té skryté vrstvy je dán i-tým prvkem tohoto pole. Výsledný objekt třídy *MLPParameters* slouží k předání všech nastavitelných vlastností najednou. Pro určení standardních hodnot parametrů MLP je zde třída *MLPConstants* se svými statickými atributy.

3.3.18 Třída Layer

Třída *Layer* reprezentuje jednu vrstvu vícevrstvé perceptronové sítě - MLP. Každá vrstva se skládá z určitého počtu neuronů. Pole těchto neuronů s názvem *perceptrons* tvoří jeden z atributů třídy *Layer*. Každé vrstvě MLP se předává vstup, který je zároveň výstupem vrstvy předchozí a na jehož základě je metodou *computeOutput* vypočítán výstup této vrstvy. Vstupy i výstupy vrstev jsou předávány v poli reálných čísel. Kromě výpočtu výstupů na vstupní signály, které se šíří dopředně, umí vrstva také vypočítat chyby, které se naopak šíří zpětně. Způsob výpočtu chyb ve skrytých a ve výstupní vrstvě se liší. Metoda *computeError* počítá chyby neuronů ve skrytých vrstvách a *computeErrorLastLayer* počítá chyby ve výstupní vrstvě. Po průchodu signálu vpřed a chyby vzad je možné upravit váhy spojení. Tuto činnost obstarává metoda *updateWeights*.

3.3.19 Třída Perceptron

Každá vrstva sítě MLP obsahuje pole referencí na své perceptrony, kterými jsou instance třídy *Perceptron*. U každého perceptronu se počítá jeho výstup na přijatý signál. Tento výstup se nejen šíří do dalších vrstev, ale využívá se i k výpočtu chyby, která připadá na daný neuron, proto se výstup v objektu neuronu uchovává prostřednictvím atributu *output*. Situace s chybou je analogická. Chyba se vypočítá a šíří na neurony v předcházejících vrstvách, ale její hodnota je použita při upravování hodnot váhových spojení, které nastává později. Proto jsem se rozhodl její hodnotu uchovat v atributu s názvem *error*. Posledním atributem třídy *Perceptron* je pole váhových spojení, přes které se výstupy a chyby šíří. Mezi metody této třídy patří pouze getry a setry atributů.

3.3.20 Třída Window

Hlavní okno aplikace je vytvořeno ve třídě *Window* z prezentační vrstvy. Tato třída dědí od *JFrame*, což jí dává možnost vytvářet grafické uživatelské rozhraní. GUI jsem se snažil vytvořit pro co možná nejjednodušší ovládání celého programu. Knihovna Swing jazyka Java nabízí možnost použití záložek jako prvku členění GUI. Záložky se používají prostřednictvím třídy *JLayeredPane* a já jsem použil pro každou kombinaci dolní a horní sítě jednu. Metoda pro tvorbu okna GUI se jmenuje *createMainWindow* a je volána z konstruktoru třídy *Window*. V *createMainWindow* jsou volány další metody, které vytvoří obsah jednotlivých záložek. Hierarchie volání metod na vytvoření celého GUI připomíná datovou strukturu strom a je zobrazena ve schématu 1 v přílohách. Načítání vstupních souborů probíhá za pomoci třídy *JFileChooser* a název načteného souboru se zobrazuje pod tlačítkem, které dialog na výběr vyvolalo. V případě, že se vybírá učící nebo testovací množina dokumentů, tedy více souborů najednou, se místo názvu souboru zobrazuje jejich počet. V záložkách příslušejících některé z kombinací sítí jsou okénka (*JTextField*), do kterých lze vyplnit všechny nastavitelné parametry. Jako počáteční hodnoty parametrů sítí ART2 resp. MLP jsou zvoleny hodnoty statických proměnných ze tříd *ART2Constants* a *MLPConstants*. Zadání jak vstupních souborů, tak i parametrů je před spuštěním zpracování zkontrolováno a pokud uživatel některou z těchto náležitostí nezadal, bude mu zobrazeno okno s pokynem k nápravě.

3.3.21 Třída ProgressBar

Dle specifikace požadavků na aplikaci v kapitole 3.2.1 má být uživatel během zpracovávání dokumentů programem informován o průběhu. K zobrazování tohoto průběhu slouží grafické okno vytvořené ve třídě *ProgressBar*, která se stejně jako třída *Window* nachází v prezentační vrstvě. Obsah okna je vytvořen v metodě *makeInside*, kde je mu přidán Border Layout Manager. Informace o názvu činnosti prováděné neuronovými sítěmi a pokyn k čekání je v centrální části layoutu v objektech třídy JLabel. Do jižní části je umístěn *JProgressBar*, který graficky znázorňuje pokrok. Aby bylo možné odhadnout, kdy bude klasifikace dokončena, ukazuje okno mimo uběhnutého času i čas zbývající. Na základě změřeného uplynulého času, množství celkové práce a množství práce vykonané lze určit odhad času zbývajícího dle vztahu 3.3.21.1.

Výpočet zbývajícího času (3.3.21.1)

$$zbývající\ čas = uplynulý\ čas\ \frac{(celkem\ práce - vykonaná\ práce)}{vykonaná\ práce}$$

Přenos informace o průběhu činnosti je realizován pomocí vzoru observable-observer, kde neuronová síť vysílá informace o průběhu a objekt třídy *ProgressBar* tyto informace zpracovává. Protože mezi přenášené informace patří název činnosti, procentuální vyjádření hotové části práce, uplynulý a zbývající čas je jako prostředek pro přenos využíváno objektů třídy *ProgressArt* resp. *ProgressMlp* pro síť ART resp. MLP. Z důvodu cizí implementace sítě SOM je pro ni využito *ProgressArt*. U MLP patří mezi přenášené informace také současná chyba. Tu jsem se rozhodl zakreslovat do grafu, který se při zpracování sítí MLP přidá do okna s průběhem.

3.3.22 Třída ErrorGraph

Graf, jenž je zobrazován během zpracovávání sítí MLP v okně s průběhem, je instancí třídy *ErrorGraph*. V grafu je znázorněn průběh chyby učení během konání učících cyklů. Z tohoto grafu lze ještě před dokončením zpracování vydedukovat, jestli spěje k dobrým výsledkům. Třída *ErrorGraph* dědí od třídy *JPanel* a překrývá metodu *paintComponent*. Díky tomu je možné vytvořit objekt GUI s vlastním obsahem. Mezi atributy třídy *ErrorGraph* patří kolekce *ArrayList* s názvem *data*, do níž jsou hodnoty chyb v průběhu učení ukládány. Data z kolekce *data* pak slouží k vykreslení křivky grafu. Hodnoty popisků na ose y grafu jsou voleny tak, aby mezi nimi byl co nejmenší krok při zachování čitelnosti. Jinými slovy tak, aby tedy nedošlo k splývání dvou popisků. Maximální hodnota popisků osy y je volena na základě maximální hodnoty chyby. Popisky osy x jsou krokovány po 50. Pokud bude vykonáno více učících cyklů než by se do grafu vešlo, objeví se posuvník, kterým lze vybrat a zobrazit libovolnou část grafu.

3.4 Nároky aplikace

Aplikace je jako celek velmi výpočetně náročná, z čehož plyne i velká náročnost časová. Výběr konkrétní kombinace sítí má zásadní vliv na dobu, kterou program pro zpracování potřebuje. Síť ART2 je ze všech použitých náročná nejméně a její trénování trvá zhruba třikrát kratší dobu než trénování SOM. U sítě MLP je výsledný čas závislý na zvolené topologii a lineárně závisí na počtu učících cyklů. Protože se časové nároky u různých sítí výrazně liší, budou konkrétní hodnoty uvedeny v kapitole 4 pro každou kombinaci sítí zvlášť. Nároky aplikace na procesor nejsou limitující, ale výkonnější procesor pochopitelně vede k rychlejšímu zpracování. Aplikace byla mimo jiné testována i na velmi starém a pomalém procesoru AMD Duron 650 Mhz, kde zpracování trvalo déle, ale bez problémů bylo dokončeno. Program pracuje s velkou množinou dat, je tedy zapotřebí mu přidělit dostatečné množství paměti. Množství paměti požadované programem závisí na počtu zpracovávaných dokumentů. Při práci s databází dokumentů od ČTK doporučuji přidělit aplikaci alespoň 200 MB paměti, což lze učinit pomocí přepínače –Xmx200M. Program je psán v Javě a je tedy nutné mít Javu na svém počítači nainstalovánu. Při vývoji a testování byla použita verze 1.6.

3.5 Hledání vhodných parametrů

3.5.1 Důležitost volby parametrů

Pro dosažení nejlepších možných výsledků klasifikace dokumentů je klíčovou volbou nejen správný typ dolní a horní sítě, ale také vhodné parametry sítí. Každý z parametrů má vliv na něco jiného a jako celek ovlivňují výsledky klasifikace. Míra ovlivnění výsledků různými parametry je různá. Protože jsou testy klasifikace velmi časově náročné a počet možných kombinací hodnot parametrů je obrovský, budou provedeny testy jen těch, o kterých

se domnívám, že mají na výsledek největší vliv. Použitá implementace sítě SOM již byla testována jejím autorem Jiřím Kůsou a já se budu nadále věnovat testování mnou implementovaných sítí, tedy ART2 a MLP.

3.5.2 Volba parametrů ART2

Jak bylo řečeno v kapitole 2.4.1, u sítí typu ART je možné kontrolovat míru podobnosti vektorů řazených do stejného shluku. Parametrem, kterým tuto podobnost může uživatel ovlivnit, je parametr ρ neboli bdělostní parametr. Čím vyšší hodnota ρ bude zvolena, tím přísněji bude na podobnost dohlíženo a vektory ve stejném shluku budou podobnější. Mechanismus, který má kontrolu podobnosti na starosti, je již zmíněný resetovací mechanismus. Hodnoty parametru je teoreticky možné volit od 0 do 1, ale prakticky pouze hodnoty od 0,7 do 1 mají smysl [FA04]. Změna parametru ρ má významný vliv na rozmístění vektorů ve shlucích a bude tedy testována. Dalším parametrem, jehož změnu je vhodné otestovat, je počet učících cyklů. Míra naučení sítě také bude výrazně ovlivňovat výsledné shlukování vektorů a bude testována. Situace s vhodným počtem učících cyklů není tak jednoduchá, jak by se zdálo. V reálném životě většinou oplatí, že čím vícekrát se něčemu člověk učí, tím lépe to pak umí, ale u ART to neplatí. Síť potřebuje více učících cyklů než jeden na své naučení, ale nesmí jich být tolik, aby nastala fáze tzv. přeučení. Výsledky klasifikace s různými hodnotami ρ a počty učících epoch budou diskutovány v kapitole 4.1 a 4.2.

3.5.3 Volba parametrů MLP

Činnost sítí MLP lze ovlivnit dvěma způsoby. Jedním je změna hodnot parametrů α a Θ, druhým je změna topologie sítě. Parametr α neboli učící poměr ovlivňuje, jak moc se budou měnit hodnoty váhových spojení po průchodu učícího vektoru, zatímco parametr Θ stejně jako u sítí ART potlačuje vzniklý šum. Změnou topologie sítě se rozumí změna počtu skrytých vrstev a počtů neuronů v nich. Testováno bude několik vybraných hodnot parametru α a taky pár vybraných topologií sítě. Změnou topologie se bude měnit oddělovací schopnost sítě, tedy schopnost sítě oddělit v prostoru vektorů jednotlivé shluky, což může pomoci pochopit, jak jsou v n-dimenzionárním prostoru, kde n je velikost vektorů, dokumenty organizovány.

4. Dosažené výsledky

V této kapitole budou diskutovány výsledky získané pomocí aplikace *Categorizator*, která byla v rámci této bakalářské práce vytvořena. Výsledky klasifikace jsou zobrazeny v tabulkách, které byly vytvořeny na základě výsledků ve výstupních souborech. Obsah tabulek je dělen na čtyři části, kde se každá týká jedné kategorie dokumentů. Část tabulky věnovaná jedné kategorii je dělena na sloupce, jejichž názvem je hodnota testovaného parametru. Hodnoty uvedené ve sloupci odpovídají procentuálnímu rozdělení dokumentů z dané kategorie a pro danou hodnotu zkoumaného parametru mezi vytvořené shluky. Hodnoty jiných parametrů, než na které je test zaměřen, jsou ponechány tak, jak je aplikace nabízí.

Z databáze ČTK bylo zapotřebí vybrat množinu trénovacích a testovacích dokumentů. Pro naučení sítí je vhodné používat vyrovnané počty dokumentů od všech kategorií, ale v databázi ČTK vyrovnané počty nejsou. Dokonce mezi prvními 6000 dokumenty se vyskytují dvě kategorie, u nichž četnost jejich dokumentů nepřesahuje dva výskyty. Dokumenty těchto kategorií jsem se rozhodl vyřadit ze zpracování. Ostatní kategorie mají vždy alespoň 40 dokumentů, a tak jsem jako trénovací množinu zvolil 160 dokumentů, kde má každá ze čtyř zpracovávaných kategorií zastoupení 40. Pro testování jsou voleny všechny dokumenty z prvních 6000 dokumentů, které patří do kategorií sport, politika, zahraničí a společnost. Zbylých 1600 dokumentů z databáze ČTK už jsou takové dokumenty, u nichž nelze jednoznačně rozhodnout o kategorii, a jsou tedy zařazeny do kategorií více. Klasifikaci takových dokumentů bych ponechal až na síti, která bude dobře fungovat pro dokumenty s jednoznačným určením kategorie.

4.1 Kombinace ART-ART

4.1.1 Kategorizace

Výsledky zpracování databáze dokumentů od ČTK takovou kombinací sítí, kde je ART jako dolní i horní síť, jsou zobrazeny v tabulce 4.1.1.1. Tato tabulka zobrazuje, jak se měnily výsledky kategorizace v závislosti na změně parametru ρ. Parametr ρ ovlivňuje rozmístění dokumentů do shluků na základě kontroly podobnosti jejich kontextových vektorů. Z tabulky 4.1.1.1 je patrné, že nebylo dle očekávání dosaženo žádané rozmístění dokumentů. Poslední řádek tabulky, který je popsán písmenem *N* označuje shluk, ve kterém se nachází nezařaditelné dokumenty, tedy takové dokumenty, u nichž parametr ρ nedovolil zařazení do žádného jiného shluku. Při zvyšování hodnoty ρ je vidět rostoucí tendence obsazenosti shluku *N*. Tuto tendenci také zobrazuje graf 4.1.1.1. Protože test změn parametru ρ neukázal žádné dobré klasifikační výsledky, nebyl ani test změn počtu učících epoch realizován. Myslím si, že důvodem špatných výsledků u ART-ART je, že spodní síť ART2 nedokáže správně vystihnout sémantiku přirozeného jazyka, čímž vznikají příliš podobné dokumentové vektory, které již není možné kategorizovat. Pokusil jsem se rozlišovací schopnost dolní sítě rozšířit zvýšením počtu shluků, které vytváří, na hodnotu 300, ale ke zlepšení výsledků nedošlo.

Zajímavým výstupem ovšem byl výpis slov řazených ke shluku č. 275, kde se zhruba ze tří čtvrtin jednalo o různá příjmení.

Tabulka 4.1.1.1 Výsledky ART-ART klasifikace – změny parametru ρ

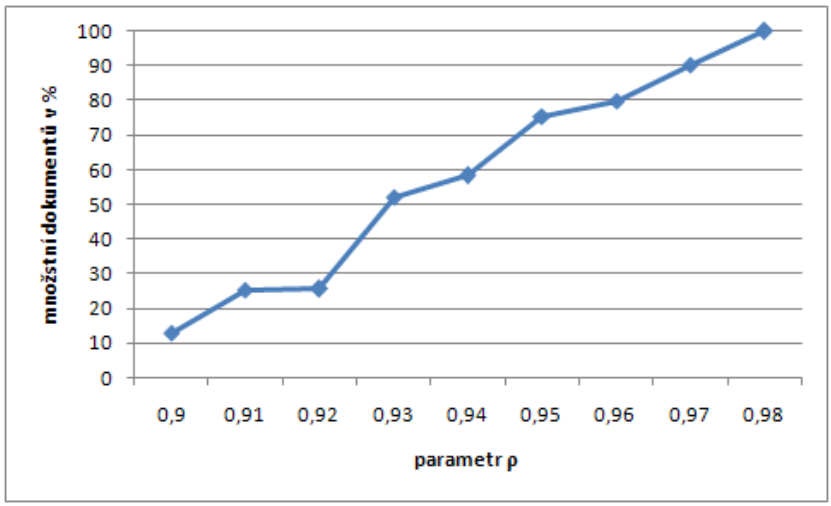
ART-ART (změny parametru ro)

Podíl dokumentů v % přiřazených danému shluku s tematickým okruhem

Číslo shluku	sport								politika								zahraniční aktualita								společnost							
	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,97	0,98	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,97	0,98	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,97	0,98	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,97	0,98
0	4,2	6	20,6	14,2	11,7	5,3	0,9	0,6	0	0,7	6	14,6	30,5	7,5	1,5	0,7	0,6	0	0,9	17,6	17,2	23,5	6,3	0	1,8	3,2	0	2,2	13,3	11,1	24,4	15,6	2,2	6,7	0	0
1	7,2	5,9	2,7	0,3	0,1	3,3	0,8	0	0	12,2	5,9	12,1	0	3,6	12	7,3	0,1	0	8,1	6,3	4,1	0	0,9	8,6	1,8	0	0	8,9	8,9	8,9	4,4	6,7	15,6	0	0	0
2	3,5	3,4	8,3	2,7	12,9	1,5	2,6	0,5	0	2,7	3,4	17,1	7,4	6	4,1	2,7	0,8	0	1,4	1,4	7,2	6,3	14,9	10	6,3	0	0	4,4	4,4	6,7	6,7	2,2	6,7	4,4	2,2	0
3	8,2	8,3	0,6	4,4	7,7	0,3	0,6	2,9	0	27,2	8,3	0	4,5	12,3	1,5	0,2	1,6	0	10,4	14	0,9	1,8	12,7	0	1,8	2,3	0	11,1	8,9	8,9	8,9	8,9	0	0	2,2	0
4	36,8	33,5	1,7	1	0,2	0,3	2,9	0,7	0	13,8	33,5	2,7	1,1	0,2	5,1	1,6	0,2	0	18,1	13,6	5,9	1,4	0	1,4	0	0,9	0	20	20	2,2	4,4	4,4	6,7	0	2,2	0
5	1,3	1	9,5	1,5	1,2	0,5	0,7	0	0	0,1	1	1,4	0,5	7,1	0,5	0,3	0	0	0,5	0,5	0,9	0	2,3	6,3	1,4	0,5	0	6,7	6,7	8,9	8,9	6,7	0	0	2,2	0
6	13,5	11,8	7,8	8,5	1,1	0,4	2,3	1,2	0	23,2	11,8	5,2	1,6	2,1	0	1,4	3,8	0	24,4	21,3	2,3	5	0,9	0,5	0,9	1,8	0	13,3	13,3	8,9	4,4	6,7	0	4,4	2,2	0
7	1,2	0,4	6,8	2,1	2,7	1,7	1,7	0,4	0	2,9	0,4	8,2	1,9	3,2	2,4	3,2	0,2	0	1,8	0,9	28,1	2,7	1,4	1,8	0,9	0,5	0	6,7	4,4	15,6	2,2	8,9	11,1	6,7	2,2	0
8	6,5	3,1	10,6	4,1	1,4	0,1	1	0,9	0	7,2	3,1	15,2	8,9	1,5	2,7	3,8	2,9	0	19	3,6	8,1	17,2	1,4	0,5	1,8	7,2	0	13,3	11,1	11,1	4,4	4,4	0	4,4	0	0
9	0,8	0,9	1,6	0,8	0,1	3,5	3,8	0,5	0	1,2	0,9	2,3	0	0,5	2,6	2,3	1,4	0	3,6	3,2	3,2	0,5	2,3	1,8	2,3	0	0	2,2	0	0	4,4	0	0	6,7	2,2	0
N	16,7	25,8	29,8	60,4	60,7	83,1	82,6	92,3	100	8,9	25,8	21,2	43,6	56	67,4	76,6	88,5	100	11,8	17,6	22,2	41,6	57	69,2	81	83,7	100	11,1	8,9	17,8	26,7	35,6	57,8	66,7	84,4	100

Ostatní parametry: 1000 učících cyklů; a=10; b=10; c=0,1; d=0,9; e=0; Θ=0,129; α=0,6; dopředné váhy=1,29; zpětné váhy=0

Graf 4.1.1.1 Výsledky ART-ART klasifikace - množství nezařazených dokumentů při změně ρ

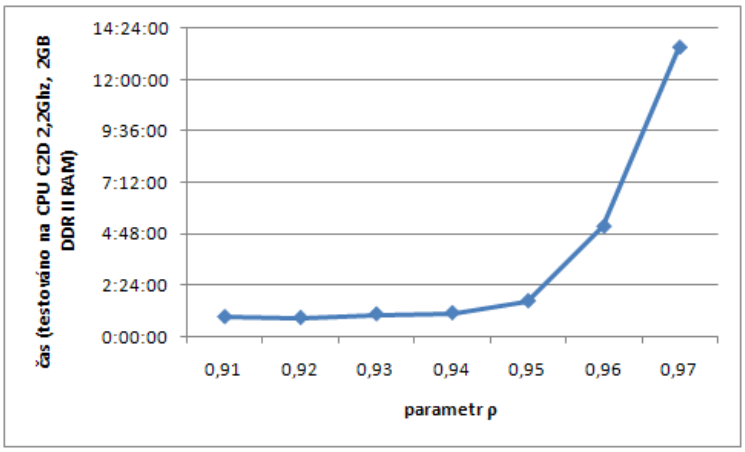


Ostatní parametry: 1000 učících cyklů; a=10; b=10; c=0,1; d=0,9; e=0; Θ=0,129; α=0,6; dopředné váhy=1,29; zpětné váhy=0

4.1.2 Časová náročnost

Zpracování dokumentů prostřednictvím ART-ART je ze všech kombinací implementovaných sítí nejméně časově náročné. Zpracování se skládá z učení sítě dolní, vytvoření dokumentových vektorů, učení sítě horní a klasifikace horní sítí. Všechny tyto fáze nějakou dobu trvají a konkrétní čas závisí na mnoha parametrech. Mezi ty nejdůležitější, co se týče vlivu na dobu zpracování, patří počet učících epoch, na němž závisí učení obou sítí lineárně, velikost množiny slov celkového slovníku resp. počet zpracovávaných dokumentů, která lineárně ovlivňuje učení sítě dolní resp. vytváření dokumentových vektorů, učení sítě horní i klasifikaci horní sítí, a hodnota parametru ρ, jenž ovlivňuje všechny zmíněné fáze. Vliv parametru ρ ovšem lineární není a zobrazuje jej graf 4.1.2.1. Z grafu je vidět, že pro hodnoty ρ do 0,95 lze získat výsledky na uvedeném stroji do dvou hodin, ale poté potřebný čas prudce stoupá. Nezbytné je ovšem dodat, že bylo zpracováváno 6000 dokumentů s použitím 160 trénovacích dokumentů při nastavení 750 učících epoch u horní i dolní sítě. Ostatním parametrům byly ponechány jejich původní hodnoty.

Graf 4.1.2.1 Výsledky ART-ART klasifikace – závislost doby zpracování na parametru ρ



Ostatní parametry: 1000 učících cyklů; a=10; b=10; c=0,1; d=0,9; e=0; Θ=0,129; α=0,6; dopředné váhy=1,29; zpětné váhy=0

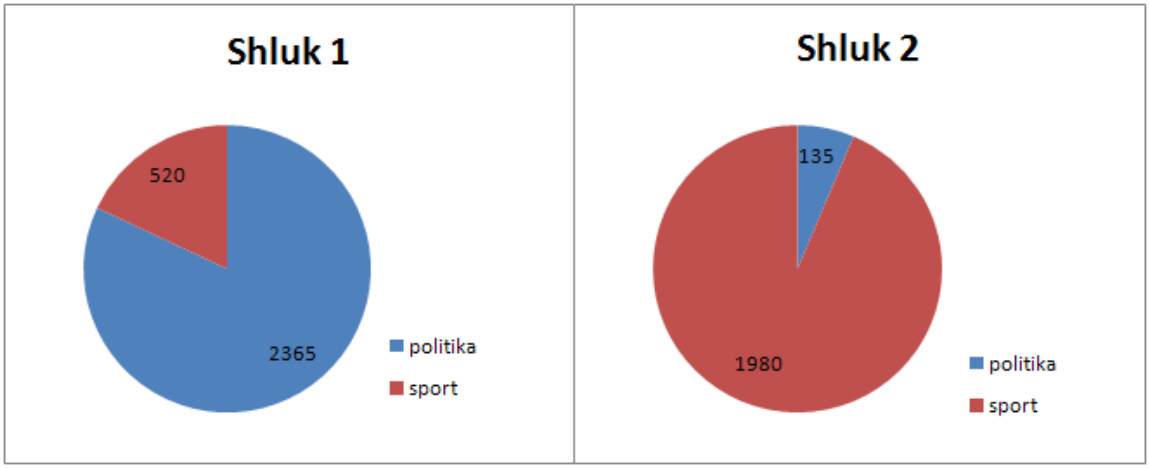
4.2 Kombinace SOM-ART

4.2.1 Kategorizace

Kombinace sítí SOM-ART přináší výhodu v podobě použití takové sítě pro zpracování slovníku všech použitých slov v databázi dokumentů od ČTK, jaká již byla úspěšně ozkoušena na dokumentech ve finštině. Navíc se jako horní síť používá taková, která nemá problémy s určením hranic mezi jednotlivými shluky. Prvním vykonaným testem bylo učení SOM-ART po 1000 učících cyklů s různými hodnotami parametru ρ. Výsledky tohoto testu jsou shrnuty v tabulce 4.2.1.1. V tabulce je vidět, že při testování nedocházelo k takovému rozptylu dokumentů do shluků, jako tomu bylo v případě ART-ART, ale dokumenty se i při vyšších hodnotách ρ držely v prvním shluku. Výjimku tvoří test pro hodnoty 0,95 a 0,98, kde k rozptylu došlo. Zajímavý je zejména test pro ρ rovno 0,95, kde se dokumentové vektory představující politický článek usadili ve shluku prvním, zatímco články sportovní ve shluku druhém. Oproti ostatním výsledkům se tato hodnota ρ u kombinace SOM-ART zdá nadějná. V tabulce 4.2.1.2 jsou výsledky testů s různým počtem učících epoch, kde parametr ρ je 0,95. Z výsledků je patrné, že při přesažení 1250 cyklů učení už dochází k přeučení sítě a není tedy vhodné takový počet cyklů volit. Nejlepších výsledků síť dosahuje mezi 750 a 1000 učícími cykly.

Jelikož je dokumentů o politice a sportu nejvíce, podařilo se dokumenty těchto kategorií oddělit nejlépe. Realizoval jsem ještě jeden test, v kterém trénovací i testovací množinu tvoří 2500 dokumentů o politice a 2500 dokumentů o sportu. Takovéto množství dokumentů databáze ČTK nabízí. Parametr ρ byl nastaven 0,95 a vykonáno bylo 750 učících cyklů. Horní síť klasifikovala dokumenty pouze do dvou shluků. Obsah shluků po klasifikaci zobrazuje graf 4.2.1.1. Z grafu je patrné, že došlo k celkem úspěšnému oddělení dokumentů z kategorií sport a politika. Dokumentů, které nebyly zařazeny do žádného ze shluků, bylo méně než 1 %.

Graf 1 Výsledky SOM-ART klasifikace – pouze politika a sport do dvou shluků



Ostatní parametry: ρ = 0,95; 750 učících cyklů; a=10; b=10; c=0,1; d=0,9; e=0; Θ=0,129; α=0,6; dopředné váhy=1,29; zpětné váhy=0

Tabulka 4.2.1.1. Výsledky SOM-ART klasifikace – změny parametru ρ

SOM-ART (změny parametru ro)

Podíl dokumentů v % přiřazených danému shluku s tematickým okruhem

Číslo shluku	sport								politika								zahraniční aktualita								společnost							
	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,97	0,98	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,97	0,98	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,97	0,98	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,97	0,98
0	100	97,6	94,8	99,5	99,9	20,5	99,5	93,8	8,4	100	98,7	99	100	100	95,9	100	100	1,2	100	99,1	98,6	100	100	82,8	99,5	99,5	2,7	100	97,8	97,8	100	100	64,4	100	100	4,4
1	0	0	4,8	0,3	0,1	77,5	0,3	1,1	4,7	0	1,3	1	0	0	4,1	0	0	10,4	0	0,9	1,4	0	0	17,2	0	0	51,1	0	2,2	2,2	2,2	0	35,6	0	0	13,3
2	0	0,5	0,4	0,2	0	0,7	0,1	5,1	0,4	0	0	0	0	0	0	0	0	2,3	0	0	0	0	0	0	0,5	0,5	1,4	0	0	0	0	0	0	0	0	2,2
3	0	1,9	0	0	0	1,3	0	0	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	14,7	0	0	0	0	0	0	0	0	5,6	0	0	0	0	0	0	0	0	0,5	0	0	0	0	0	0	0	0	0	8,9
5	0	0	0	0	0	0	0	0	5,8	0	0	0	0	0	0	0	0	58,3	0	0	0	0	0	0	0	0	10,4	0	0	0	0	0	0	0	0	0	44,4
6	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0,5	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	56,3	0	0	0	0	0	0	0	0	14,7	0	0	0	0	0	0	0	0	16,3	0	0	0	0	0	0	0	0	0	13,3
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2,9	0	0	0	0	0	0	0	0	1,8	0	0	0	0	0	0	0	0	0	4,4
9	0	0	0	0	0	0	0	0	3,7	0	0	0	0	0	0	0	0	3,6	0	0	0	0	0	0	0	0	13,1	0	0	0	0	0	0	0	0	0	8,9
N	0	0	0	0	0	0	0	0	4,7	0	0	0	0	0	0	0	0	1,1	0	0	0	0	0	0	0	0	2,3	0	0	0	0	0	0	0	0	0	0

Ostatní parametry: 1000 učících cyklů; a=10; b=10; c=0,1; d=0,9; e=0; Θ=0,129; α=0,6; dopředné váhy=1,29; zpětné váhy=0

Tabulka 4.2.1.2 Výsledky SOM-ART klasifikace – změny počtu učících

SOM-ART (změny počtu učících cyklů)

Podíl dokumentů v % přiřazených danému shluku s tematickým okruhem

Číslo shluku	sport									politika										zahraniční aktualita										společnost									
	1	10	100	500	750	1000	1250	1500		1	10	100	500	750	1000	1250	1500			1	10	100	500	750	1000	1250	1500			1	10	100	500	750	1000	1250	1500		
0	96,3	99,4	88,4	99,3	7,5	99,5	99,7	99,2		100	100	99,9	100	69,3	100	100	100			99,5	100	97,3	100	59,3	100	100	100			100	100	100	100	46,7	100	100	100		
1	3,7	0,3	6,4	0,7	1,1	0,1	0,1	0,8		0	0	0	0	0	0	0	0			0,5	0	0,9	0	0	0	0	0			0	0	0	0	0	0	0	0		
2	0	0,3	0,1	0	89,7	0,4	0,2	0		0	0	0	0	30,6	0	0	0			0	0	0	0	40,7	0	0	0			0	0	0	0	53,3	0	0	0		
3	0	0	0,2	0	1,7	0	0	0		0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0		
4	0	0	5	0	0	0	0	0		0	0	0,1	0	0	0	0	0			0	0	1,8	0	0	0	0	0			0	0	0	0	0	0	0	0		
5	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0		
6	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0		
7	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0		
8	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0		
9	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0		
N	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0		

Ostatní parametry: ρ=0,95; a=10; b=10; c=0,1; d=0,9; e=0; Θ=0,129; α=0,6; dopředné váhy=1,29; zpětné váhy=0

4.2.2 Časová náročnost

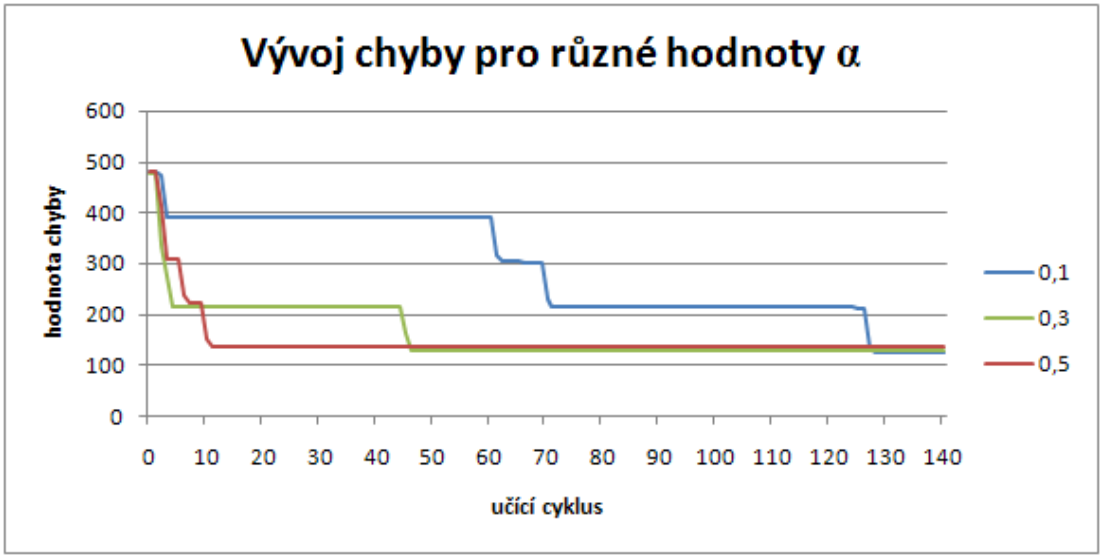
SOM-ART je výpočetně náročnější soustava sítí než ART-ART. Důvodem je právě dolní síť SOM, jejíž učení trvá mnohem déle než učení sítě ART. Na počítači s procesorem C2D 2,2 Ghz a 2GB DDR II RAM trvá zpracování kolem šesti hodin, přičemž faktory horní sítě, které ovlivňovaly dobu zpracování u kombinace ART-ART (viz 4.1.2) zde platí také, protože i zde je ART2 horní sítí.

4.3 Kombinace SOM-MLP

4.3.1 Kategorizace

SOM-MLP stejně jako SOM-ART využívá pro vytváření dokumentových vektorů síť SOM, ale učení sítě, která provádí klasifikaci dokumentových vektorů, probíhá s učitelem. Při testování kombinace SOM-MLP byly měněny vlastnosti MLP. Nejprve jsem provedl testy klasifikace s různými hodnotami koeficientu učení α, který určuje míru naučení sítě učícím vektorem. Výsledky tohoto testu s hodnotami α 0,1, 0,3 a 0,5 jsou zobrazeny v tabulce 4.3.1.1. Jak učící poměr ovlivňuje vývoj chyby učení během konání učících cyklů, ukazuje graf 4.3.1.1. Z grafu je vidět, že při zvýšení hodnoty koeficientu učení chyba klesá rychleji, ale zastaví se na vyšší hodnotě než u učení s menší hodnotou α. Dalším testem, který jsem provedl, byl test klasifikace při různém počtu neuronů v první a druhé skryté vrstvě sítě MLP. V kapitole 2.6.2 je diskutován výběr správného počtu neuronů ve skrytých vrstvách na základě velikosti vstupních vektorů a počtu klasifikačních tříd. S použitím rovnic 2.6.2.1 a 2.6.2.2 dostáváme, že první skrytá vrstva má mít 71 a druhá 17 neuronů. Tento doporučený počet byl testován spolu s počtem vyšším i nižším v obou vrstvách. Dosažené výsledky zobrazené v tabulce 4.3.1.2 nejsou uspokojivé, protože vždy došlo k naučení sítě pouze jednou kategorií dokumentů a k neuronu tuto kategorii představující byly během klasifikace přiřazeny dokumenty všechny.

Graf 4 Vývoj chyby při SOM-MLP klasifikaci s různými hodnotami α



Ostatní parametry: 71 neuronů v první a 17 v druhé skryté vrstvě

Tabulka 4 Výsledky SOM-MLP klasifikace – změny parametru α

SOM-MLP (změny parametru alfa)												
Název shluku	Podíl dokumentů v % přiřazených danému shluku s tematickým okruhem											
	sport			politika			zahraničí			společnost		
	0,1	0,3	0,5	0,1	0,3	0,5	0,1	0,3	0,5	0,1	0,3	0,5
sport	0	0	0	0	0	0	0	0	0	0	0	0
politika	100	0	0	100	0	0	100	0	0	100	0	0
zahraničí	0	0	0	0	0	0	0	0	0	0	0	0
společnost	0	100	100	0	100	100	0	100	100	0	100	100

Ostatní parametry: 10000 učících cyklů; 71 neuronů v první a 17 v druhé skryté vrstvě

Tabulka 5 Výsledky SOM-MLP klasifikace – změny počtu neuronů v 1. a 2. skryté vrstvě

SOM-MLP (změny topologie)												
Název shluku	Podíl dokumentů v % přiřazených danému shluku s tematickým okruhem											
	sport			politika			zahraničí			společnost		
	40-4	71-17	100-20	40-4	71-17	100-20	40-4	71-17	100-20	40-4	71-17	100-20
sport	0	0	0	0	0	0	0	0	0	0	0	0
politika	100	0	100	100	0	100	100	0	100	100	0	100
zahraničí	0	0	0	0	0	0	0	0	0	0	0	0
společnost	0	100	0	0	100	0	0	100	0	0	100	0

Ostatní parametry: 10000 učících cyklů; α=0,1

4.3.2 Časová náročnost

Doba potřebná pro zpracování dokumentů kombinací SOM-MLP je součtem časů, který spotřebuje dolní síť SOM a horní síť MLP. SOM při učení celkovému slovníku databáze dokumentů ČTK potřebuje zhruba 5 hodin (na počítači s procesorem C2D 2,2 Ghz a 2GB DDR II RAM). Doba činnosti sítě MLP závisí lineárně na počtu učících cyklů, přičemž 30000 cyklů se 160 učícími vektory trvá kolem jedné hodiny.

5. Závěr

Hlavním úkolem této práce bylo zamyslet se nad kategorizací česky psaných textových dokumentů a pokusit se jí realizovat pomocí umělých neuronových sítí. Prvním krokem ke splnění vytyčeného cíle bylo najít způsob, jak vytvořit reprezentaci dokumentu, která bude odrážet jeho obsah. Při procházení materiálů zabývajících se zpracováním sémantiky přirozeného jazyka jsem narazil na publikaci vytvořenou na Helsinki University of Technology Timo Honkelou. Autor zde popisuje, jak lze klasifikovat textové dokumenty ve finštině a angličtině. Tento způsob zpracování dokumentů jsem se rozhodl vyzkoušet na dokumentech v českém jazyce.

Na univerzitě v Helsinkách použili pro klasifikaci textových dokumentů dvouúrovňovou architekturu neuronových sítí, kde první (dolní) vrstva shlukuje slova a vytváří reprezentace dokumentů (tzv. dokumentové vektory), zatímco vrstva druhá (horní) shlukuje dokumenty. Tuto architekturu sítí jsem převzal a použil jí ve své práci. Finové použili pro dolní i horní vrstvu sít SOM. Já jsem pro dolní vrstvu zvolil mimo jiné také síť SOM, protože si myslím, že je k shlukování slov vhodná. Požadavky kladené na síť z horní vrstvy, které říkají, že dokumenty mají být klasifikovány do omezené množiny klasifikačních tříd, mě vedly k volbě odlišných sítí, konkrétně ART2 a MLP. U těchto sítí totiž není problém určit hranice mezi vytvořenými shluky (podrobněji viz 3.2.7). Pro úplnost informace o výběru konkrétních sítí chybí jen dodat, že síť ART2 byla otestována i jako síť dolní vrstvy.

Rozhodl jsem se nevytvořit pouze implementace výše zmíněných sítí, ale naprogramoval jsem aplikaci, která implementace všech sítí obsahuje a umožňuje tím uživateli jednotlivé kombinace lépe testovat a porovnávat dosažené výsledky. Při tvorbě aplikace jsem se soustředil na její modulárnost, aby do ní bylo možné jednoduše přidávat další implementace sítí. Výslednou aplikaci jsem pojmenoval *Categorizator* a její převážná část je mé vlastní dílo. Pouze implementace sítě SOM je převzata z bakalářské práce Jiřího Kůsy z roku 2007.

Pomocí aplikace *Categorizator* jsem provedl velké množství testů. Hodnoty nastavitelných parametrů sítí výrazně ovlivňují výsledky těchto testů, a tak jsem se snažil nalézt takovou kombinaci hodnot parametrů, která bude vykazovat nejlepší možné výsledky. Výsledky provedených testů s vybranými hodnotami parametrů jsou diskutovány v kapitole 4. V souhrnu lze říci, že nejlepších výsledků dosahuje taková kombinace sítí, kde SOM tvoří dolní a ART2 horní vrstvu. U této kombinace se podařilo od sebe oddělit dokumenty o politice a sportu na zhruba 85 %. Věřím tomu, že by toto číslo bylo možné ještě navýšit. Naopak nejhorší výsledky podává kombinace ART-ART, kde se dokumenty rozptylují zcela náhodně a o klasifikaci se zde hovořit nedá. Důvodem proč testy SOM-ART dopadly mnohem lépe než testy ART-ART vidím v tom, že dolní vrstva tvořená sítí SOM dokáže lépe podchytit kontext slov v českém jazyce a vlastnost sítě ART, tedy jednoznačnost určení hranic shluků, která je využita pro síť horní vrstvy, je u sítě z dolní vrstvy spíše omezující.

Vzhledem k časové náročnosti testů nebylo možné vyzkoušet všechny kombinace hodnot parametrů a myslím si, že dalším testováním by se dalo dosáhnout ještě lepších výsledků. Jako další rozšíření této práce je také možné využít modularity vytvořené aplikace a doplnit ji o další typy neuronových sítí. I zvolený popis slov pomocí kontextových vektorů by bylo možné modifikovat, protože byl původně navržen pro angličtinu, kde je ale mnohem pevnější slovosled než v jazyce českém.

Přehled zkratek

ČTK	Česká tisková kancelář
ART	Adaptive Resonance Theory
SOM	Self-Organizing Map
MLP	Multilayer Perceptron
LAM	Linear Associative Memory
BP	Back-Propagation
GUI	grafické uživatelské rozhraní
WEBSOM	SOM architektura pro zpracování dokumentů dostupná přes web

Literatura

[AL97] Almeida Luis B. *Multilayer perceptrons* In: E. Fiesler, R. Beale. Handbook of neural computation. IOP Publishing Ltd and Oxford University Press, 1997.

[IIR07] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2007.

[FA94] Fausett Laurene. *Fundamentals of neural networks*. Prentice-Hall, New Jersey, 1994.

[FI97] Fiesler Emile. *Neural Network Topologies* In: E. Fiesler, R. Beale. Handbook of neural computation. IOP Publishing Ltd and Oxford University Press, 1997.

[HV06] Hnízdilová Veronika. *Diplomová práce: Sémantika jazyka a Kohonenova mapa*. Plzeň, 2006.

[HO97] Honkela Timo. *Self-Organising Maps in natural language processing*. Helsinki University of Technology, 1997.

[KL07] Krčmář Lubomír. *Bakalářská práce: Vytváření vstupních vektorů pro WEBSOM.* Plzeň, 2007.

[KU93] Kung Sun Yuan. *Digital neural networks*. Prentice-Hall, New Jersey, 1993.

[NO97] Noyes James L. *Neural Network Training* In: E. Fiesler, R. Beale. Handbook of neural computation. IOP Publishing Ltd and Oxford University Press, 1997.

[ŠM04] Šnorek Miroslav. *Neuronové sítě a neuropočítače*. Praha, 2004.

[VM] Veselovský Michal. *Neuronové sítě.* http://www.kiv.zcu.cz/studies/predmety/uir/NS/NS.html

[WIKCJ] Wikipedie. *Portál: Český jazyk.* http://cs.wikipedia.org/wiki/Portál:Český_jazyk

Přílohy

Seznam příloh

Algoritmus 1 Učení ART

```
for (počet učících epoch) {
    for (všechny učící vektory){
        uprav aktivace v F1 s daným vstupním vektorem;
        reset = true;
        vítěz = null;
        while(reset == true){
            winner = najdi vítěze v F2;
            uprav aktivace v F1;
            r = spočítej hodnotu resetu;
            if(r < (ro - e)){
                zablokuj vítězný shluk;
            }
            else{
                reset = false;
            }
        }
        uprav váhy vítěze;
        povol všechny zablokované v F2;
        vynuluj výstupy v F1;
    }
}
```

Algoritmus 2 Vybavování ART

```
for (všechny testovací vektory){
    uprav aktivace v F1 s daným vstupním vektorem;
    reset = true;
    vítěz = null;
    while(reset == true){
        winner = najdi vítěze v F2;
        uprav aktivace v F1;
        r = spočítej hodnotu resetu;
        if(r < (ro - e)){
            zablokuj vítězný shluk;
        }
        else{
            reset = false;
        }
    }
    přidej vítězi popisek vektoru;
    povol všechny zablokované v F2;
    vynuluj výstupy v F1;
}
```

Algoritmus 3 Učení MLP

```
for (počet učících epoch) nebo while(celková chyba < max. chyba){
        celková chyba = 0;
        for (všechny učící vektory) {
                dopředné šíření vstupu;
                zpětné šíření chyby;
                celková chyba = celková chyba + současna chyba
        }
}
```

Algoritmus 4 Vybavování MLP

```
for (všechny testovací vektory) {
        dopředné šíření vstupu;
        najdi vítězný neuron;
}
```

Schéma 1 Tvorba obsahu hlavního okna GUI

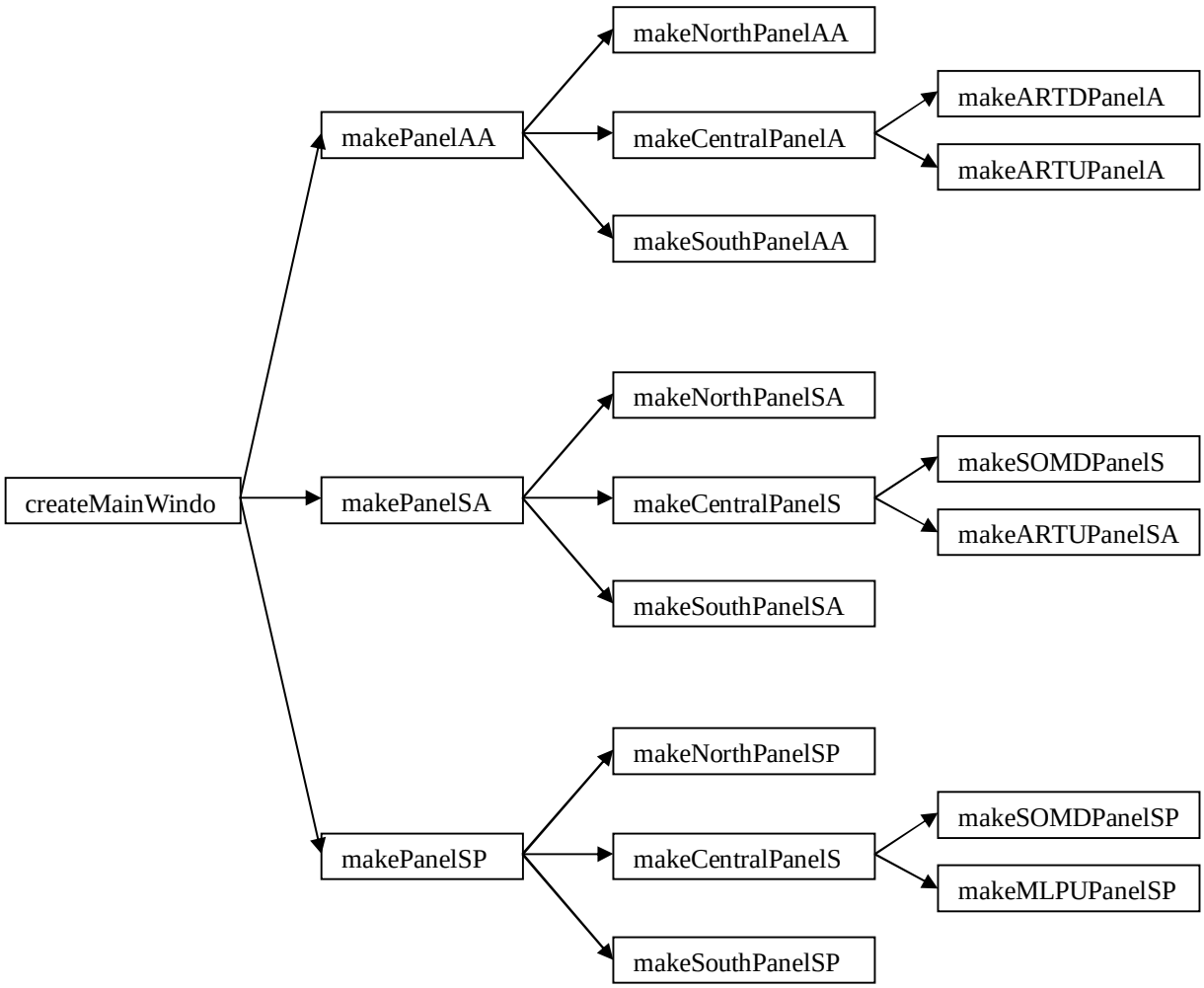


Schéma 2 Diagram činnosti klíčové metody workOut platný pro třídy ART_ART, SOM_ART a SOM_MLP

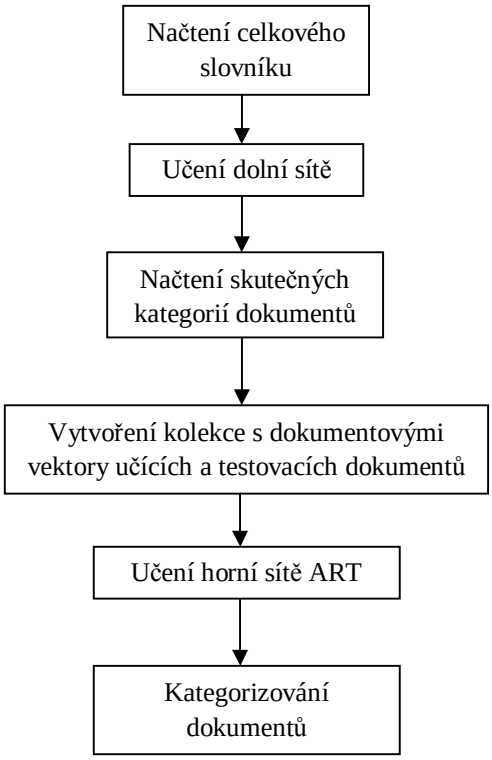


Schéma 3 UML diagram aplikace

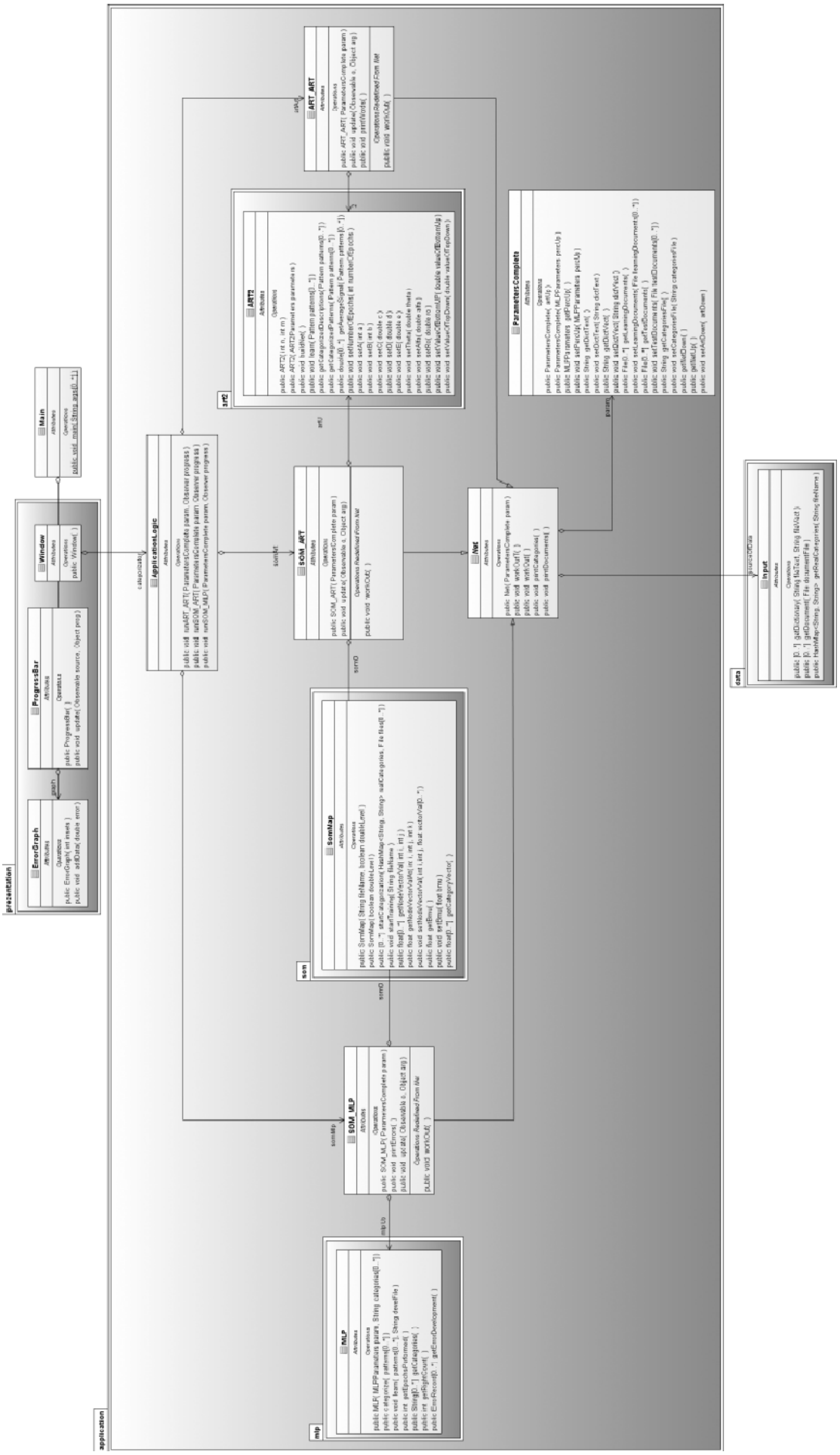


Schéma 4 UML diagram balíku art

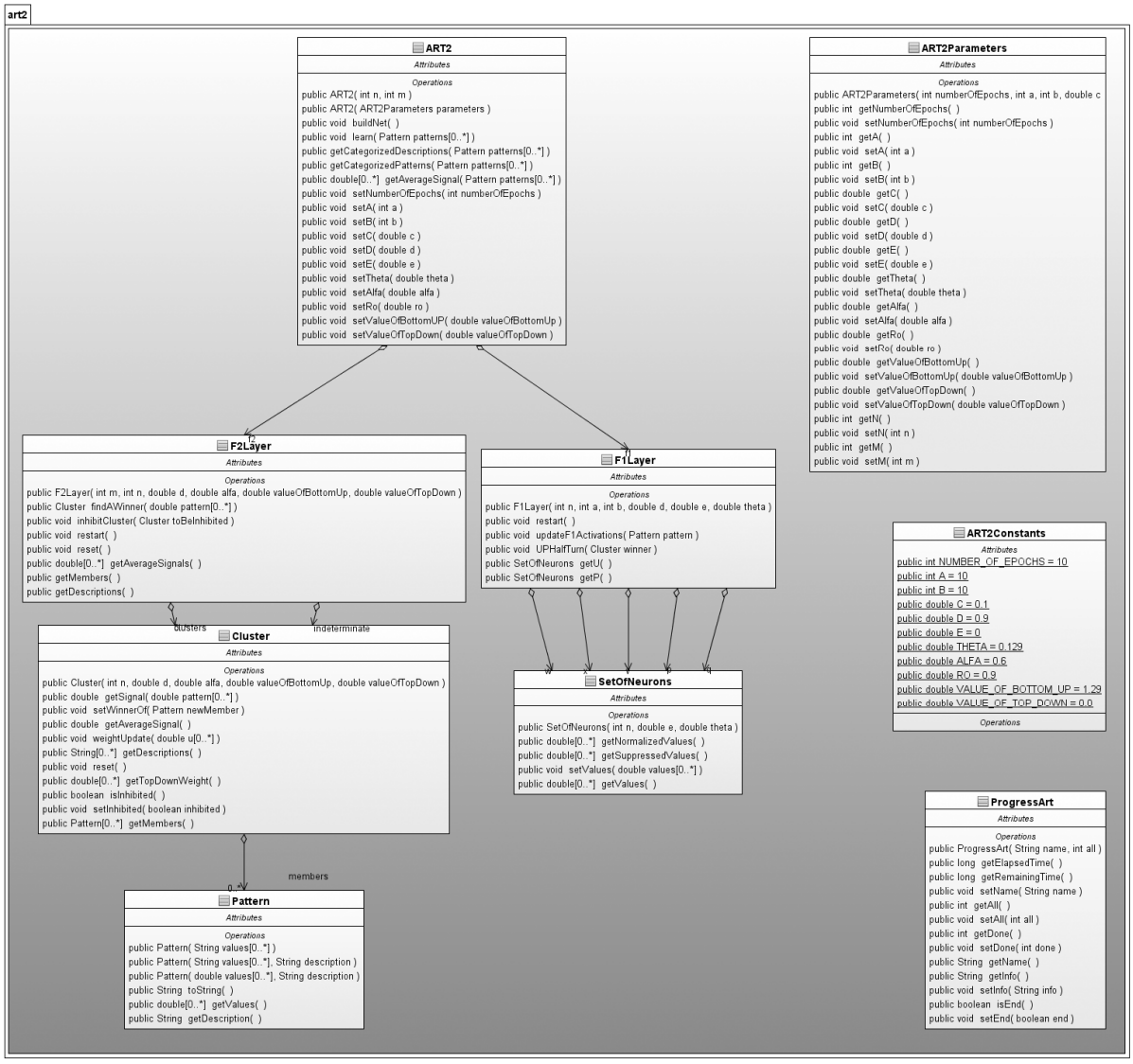


Schéma 5 UML diagram balíku som

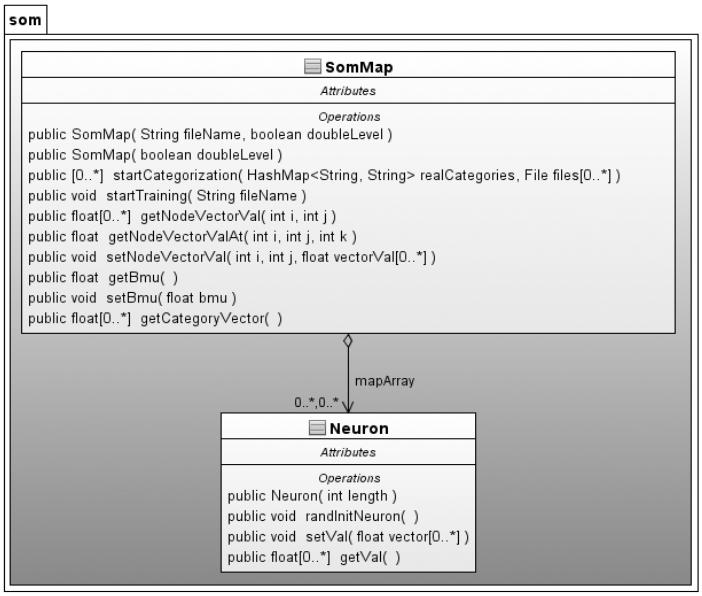
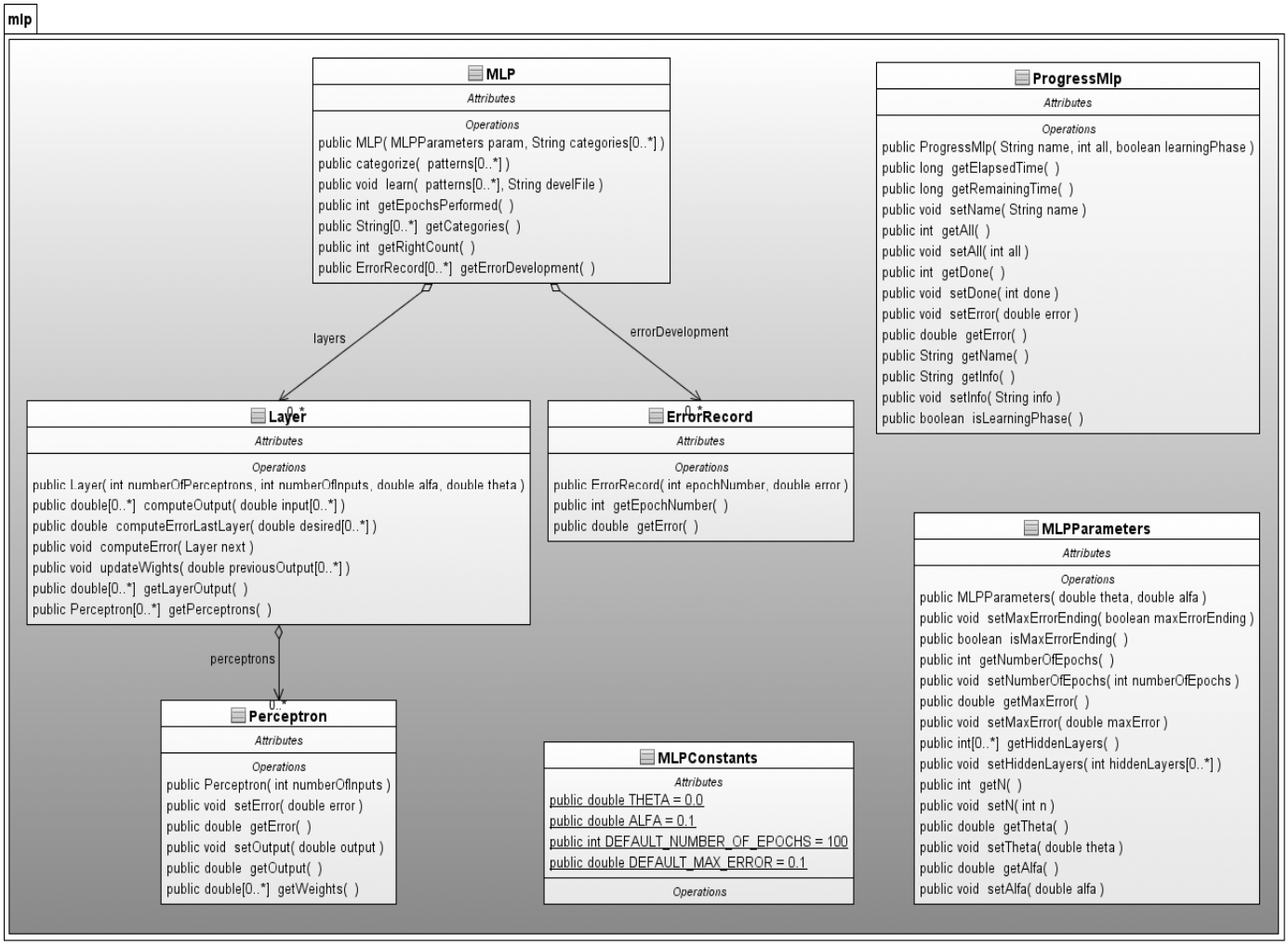


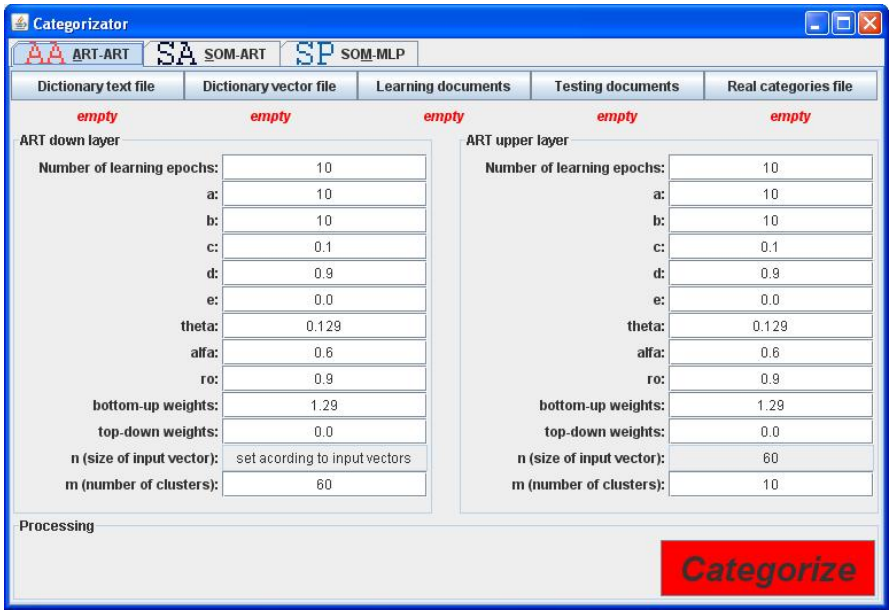
Schéma 6 UML diagram balíku mlp



Uživatelská příručka

Aplikace Categorizator slouží ke kategorizování dokumentů třemi rozdílnými metodami. Každá z těchto metod podává jiné výsledky a to hlavně na základě zadaných parametrů. Před spuštěním tohoto programu je doporučeno nainstalovat JDK 1.6 pokud ho ještě na svém počítači nainstalované nemáte. Dále budeme potřebovat textové soubory, které lze získat pomocí bakalářské práce Lubomíra Krčmáře. Program je náročný jak na výpočetní výkon počítače, tak na paměť. Pro zpracování velkého počtu dokumentů (v řádech tisíců) je doporučeno aplikaci přidělit větší množství paměti přepínačem –Xmx.

Po spuštění aplikace se Vám uprostřed obrazovky objeví hlavní okno celého programu, které je zobrazeno na obr. 1.



Obr. 1 Hlavní okno aplikace – záložka ART-ART

Toto grafické uživatelské rozhraní je členěno na záložky. Hlavní okno obsahuje tři záložky a každá reprezentuje jednu metodu kategorizování textových dokumentů, tedy jednu kombinaci dolní a horní sítě. Následující kapitoly Vás provedou používáním každé kombinace.

ART-ART

Metoda ART-ART je zastoupena záložkou první. Než bude možné zpracování dokumentů spustit, je zapotřebí učinit několik kroků, které budou nyní popsány.

Načítání vstupních souborů

Nejvýše na této záložce můžeme vidět pět tlačítek, která nám umožní načíst všechny potřebné vstupní soubory. Tlačítko první je *Dictionary text file*. Kliknutím na něj vyvoláme zobrazení dalšího okna, v kterém je možné procházet všechna dostupná místa v našem počítači a najít zde textový soubor, jenž obsahuje všechna slova ze všech dokumentů, které chceme kategorizovat. Druhé tlačítko nese název *Dictionary vector file*. Po jeho stisknutí se

nám objeví stejné okno jako pro předchozí tlačítko, ale nyní budeme vybírat textový soubor, ve kterém se nachází slovní vektory každého slova celkového slovníku, tudíž toho souboru, který jsme načítali v předchozím kroku. Třetím tlačítkem je *Learning documents* a jak už název napovídá, půjde o načítání množiny učících dokumentů. Po stisknutí tlačítka se objeví již dobře známé okénko pro výběr souborů, ale tentokrát je umožněno vybírat více souborů najednou. Výběr souborů záleží čistě na uživateli, jen je nutné, aby všechny soubory měly za svým názvem písmeno *v*, které určuje, že obsahem souborů už nejsou dokumenty v textové podobě, ale jednotlivá slova dokumentů byla nahrazena kontextovými vektory. Následující tlačítko s názvem *Testing documents*, funguje obdobně. Jen se nyní do programu načítá testovací množina dokumentů. Dokumenty z této množiny budou po naučení sítí klasifikovány a výsledky klasifikace zobrazeny. Poslední tlačítko je popsáno *Real categories file* a slouží k načítání souboru, který obsahuje skutečné, tedy člověkem určené, kategorie dokumentů. Po každém načtení, které jsme pomocí pěti tlačítek v horní části záložky provedli, se objevil zelený výpis. Výpis je umístěn vždy pod tlačítkem, které načítání vyvolalo, a zobrazuje informaci o stavu. Při načítání samostatných souborů zobrazuje jejich název, při načítání množin ukazuje, kolik souborů množina obsahuje. Lze si tedy zkontrolovat, jestli nedošlo k omylu.

Nastavování parametrů

Další částí je nastavování parametrů sítí ART. Metoda ART-ART se skládá ze dvou ART sítí, horní a dolní. Parametry obou sítí lze nastavit. Už předem jsou všechna políčka vyplněna implicitními hodnotami, ale téměř každá z těchto hodnot lze změnit. Jedinou výjimkou je parametr ‘n’, jenž představuje délku vstupního vektoru. Délka vstupního vektoru dolní vrstvy ART je nastavena na základě vstupního souboru se slovními vektory a délka vstupního vektoru vrstvy horní je zase závislá na počtu neuronů ve vrstvě dolní, tudíž se na základě změny této hodnoty mění. Podrobný popis každého z parametrů a také jeho dovolené rozsahy hodnot lze nalézt v knize Fausett, L.: Fundamentals of Neural Networks, Prentice-Hall, New Jersey, 1994 na stránkách 218-287. Dle tohoto materiálu jsou také parametry pojmenovány a se stejnými jmény funguje celý program. Stručný výpis parametrů s jejich dovolenými rozsahy naleznete v tabulce 1. Význam a interval dovolených hodnot parametrů je pro obě vrstvy stejný.

Označení	Název/Význam	Interval povolených hodnot
number of learning epochs	počet učících epoch	<1 ; ∞)
a	a	<1 ; ∞)
b	b	<1 ; ∞)
c	c	(0 ; 1)
d	d	(0 ; 1)
e	e, brání dělení nulou	<0 ; ∞)
theta	parametr potlačování šumu	<0 ; ∞)
alfa	učící poměr (rychlost učení)	(0 ; 1)
ro	bdělostní parametr	<0,7 ; 1)
bottom-up weights	dopředné váhy	<0 ; ∞)
top-down weights	zpětné váhy	<1 ; ∞)
m	počet shluků	<1 ; ∞)

Tabulka 1 Parametry ART

Zpracování

V dolní části záložky ART-ART je umístěn blok s názvem *Processing*, ve kterém se nachází velké červené *Categorize* tlačítko. Jeho stiskem dáme programu pokyn ke zpracování zadaných vstupních souborů s uvedenými parametry. Během činnosti programu, protože může být v závislosti na velikosti zpracovávané množiny časově náročná, je zobrazeno malé okénko s nadpisem *Progress*. Úkolem tohoto okénka je zobrazovat stavové informace činnosti programu. Bude zobrazen název prováděné činnosti a procentuální vyjádření již vykonané části. Na závěr zpracování program informuje o dokončení své činnosti informačním oknem.

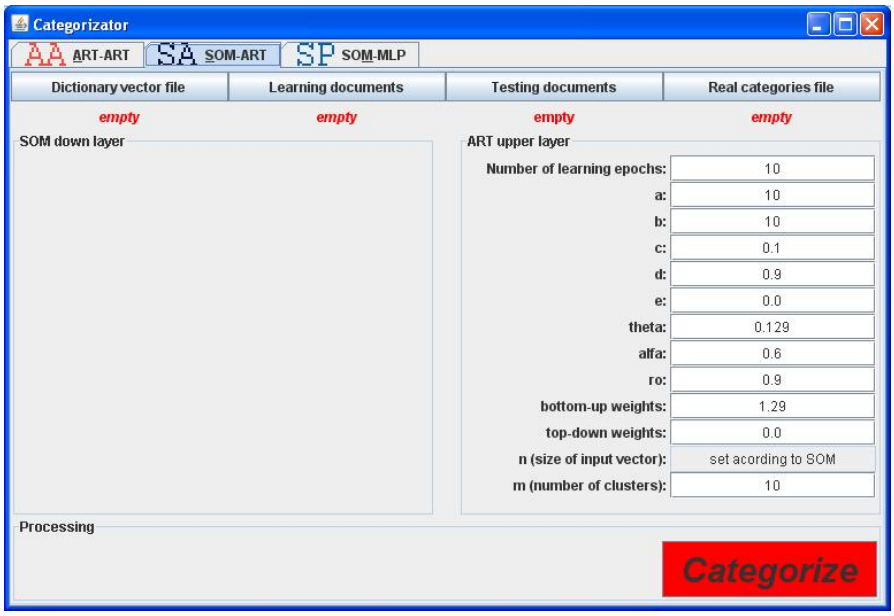
Výstup

Výsledky provedené kategorizace se nacházejí v adresáři na tom samém místě, kde byla aplikace spuštěna. Název adresáře je *Results yyyy-m-d h-m-s*, kde za klíčovým slovem *Results* je výpis data a času začátku kategorizace. Adresář obsahuje několik textových souborů. Soubory, jejichž název začíná slovem *Cluster*, které je následováno číslem, obsahují všechna slova, která byla zařazena do stejného shluku. Shluky neboli clustery jsou identifikovány čísly. Dalším textovým souborem je *Document vectors*, kde lze najít dokumentový vektor každého kategorizovaného dokumentu i s jeho skutečnou kategorií. Posledním souborem je soubor *Results yyyy-m-d h-m-s.txt*, kde datum a čas udává, kdy byla kategorizace dokončena. Tento soubor obsahuje všechny parametry, se kterými obě sítě ART pracovaly a výpis obsahu každého shluku kategorií. Shlukem se rozumí jeden cluster horní sítě ART. Výpis byl zvolen tak, aby i když do aplikace vložíme různé počty různých kategorií, dostaneme smysluplné a nezkreslené výsledky. Za číselným označením každého clusteru následuje procento a název kategorie. Název kategorie je vzat ze souboru se skutečnými kategoriemi, který se načítal tlačítkem *Real categories file*. Procento, které tomuto názvu předchází, bylo získáno jako podíl počtu výskytů dokumentů z dané kategorie na tomto clusteru děleno celkovým počtem dané kategorie v celé databázi, samozřejmě vynásobeno stem, abychom dostali procenta. Tento výpočet pak dobře ukáže, kolik procent dokumentů z nějaké kategorie se řadí do jednotlivých clusterů. Pokud by se ve výsledcích objevila čísla přesahující 90 % v odlišných clusterech, pak by se dalo hovořit o úspěšné klasifikaci dokumentů. Pokud ale máme všude jen procenta malá nebo velká ale ve stejných

clusterech, pak se zřejmě síti nepodařilo podchytit podobnost dokumentů ze stejné kategorie a odlišit je od kategorií jiných.

SOM-ART

Způsob zpracování SOM-ART lze vybrat kliknutím na druhou záložku v hlavním okně. Na obrazovce uvidíme okno z obr 2, kde lze nastavit vše potřebné pro kategorizaci dokumentů metodou SOM-ART.



Obr. 2 Záložka SOM-ART

Načítání vstupních souborů

V horní části okna jsou opět tlačítka pro výběr vstupních souborů. Nyní zde chybí první tlačítko pro výběr celkového slovníku, není zde třeba. Ostatní tlačítka fungují stejně jako u metody ART-ART a je také třeba všechny, zde čtyři, načítání provést.

Nastavování parametrů

Nastavení parametrů sítě SOM dovoleno není, ale parametry horní sítě ART lze nastavit a to opět stejně jako u předchozí metody. Opět se nevyplňuje parametr *n*, který značí délku vstupních vektorů. Tuto činnost za nás vykoná program sám a tuto délku určuje na základě počtu neuronu vytvořených v síti SOM. Dovolené rozsahy ostatních parametrů se řídí tabulkou, která se nachází u návodu na ART-ART v části nastavování parametrů.

Zpracování

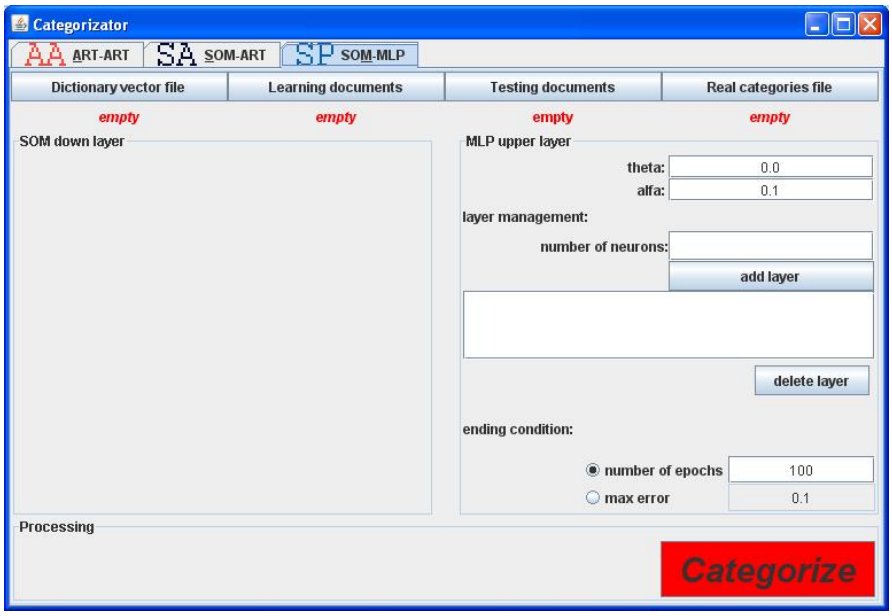
Zpracování se spouští tlačítkem *Categorize*. O průběhu zpracování stejně jako u kombinace ART-ART informuje okno *Progress*.

Výstup

Výsledky najdeme opět v adresáři pojmenovaném stejně jako u metody ART-ART, tedy *Results yyyy-m-d h-m-s*. To, že jsme kategorizovali jinou metodou, poznáme hned po otevření souboru s výsledky, který je také ve formátu *Results yyyy-m-d h-m-s.txt*, kde je na první řádce název použité metody napsaný. Na dalších řádkách se nachází výpis všech nastavených parametrů a na závěr výpis obsahů clusterů, který má stejný formát jako u metody ART-ART. Dokumentové vektory všech zpracovávaných dokumentů se nacházejí v souboru *Document vectors.txt*.

SOM-MLP

Třetí – poslední – záložka skýtá možnost zpracování dokumentů takovou kombinací sítí, kde SOM tvoří dolní a MLP horní síť. Záložka vypadá tak, jak ji zobrazuje obr 3.



Obr. 3 Záložka SOM-MLP

Načítání vstupních souborů

Tlačítka, která dovolí načtení příslušných vstupních souborů, jsou opět situována na horním okraji záložky. Jsou to ta samá tlačítka, kterými se načítá vstup u kombinace SOM-ART.

Nastavování parametrů

U sítě SOM se parametry opět nenastavují, nicméně síť MLP prostor pro nastavitelné parametry nabízí. Lze nastavit hodnotu parametru Θ a α. Zatímco Θ je parametr potlačování šumu, α udává učící poměr. Mimo nastavení těchto parametrů lze ještě modifikovat topologii sítě určením počtu skrytých vrstev a počtu neuronů v nich. Nová skrytá vrstva s žádaným počtem neuronů se přidá zadáním počtu neuronů do políčka s popiskem *number of neurons*

a stiskem tlačítka *add layer*. Takto lze přidávat libovolné množství vrstev s libovolným počtem neuronů. Pro odebrání vrstvy ji stačí ve výpisu označit a stisknout tlačítko *delete layer*. Dále je možné si zvolit ukončovací podmínku. Lze ukončit zpracovávání po vykonání daného počtu učících cyklů, nebo po dosažení určené hodnoty chyby učení. Při výběru *number of epochs* se na hodnotu v poli *max error* nebere zřetel, ale při výběru *max error* slouží hodnota v poli *number of epochs* jako limitní počet epoch, po kterém se učení ukončí, i když chyby dosaženo nebylo. Nestane se tedy, že pokud nelze chyby dosáhnout, tak program nikdy neskončí.

Zpracování

Po zadání všech vstupních souborů lze spustit zpracovávání tlačítkem *Categorize*. O průběhu zpracování stejně jako u kombinace ART-ART a SOM-ART informuje okno *Progress*. Až dojde na učení sítě MLP, bude v okně *Progress* zobrazen graf s průběhem chyby učení.

Výstup

Výsledky obsahuje adresář *Results yyyy-m-d h-m-s*, s konkrétním datem a časem, kdy byla kategorizace započata. Tento adresář obsahuje soubor *Document vectors* stejně jako u všech metod zpracování. Soubor s výsledky s názvem začínajícím slovem *Results* má podobný obsah jako u sítí ART-ART a SOM-ART. Na jeho počátku jsou vypsány nastavené parametry a také informace o skrytých vrstvách. Síť MLP si vytváří tolik shluků, do kolika kategorií bude dokumenty klasifikovat a na rozdíl od ART-ART a SOM-ART jsou shluky rovnou pojmenovány dle kategorie dokumentů, které do daného shluku mají být řazeny. Výstupními soubory, které se u jiných kombinací sítí nenacházejí, jsou *errors.txt a outputs.txt*, přičemž první jmenovaný obsahuje vývoj chyby a druhý vývoj výstupů v průběhu učení.